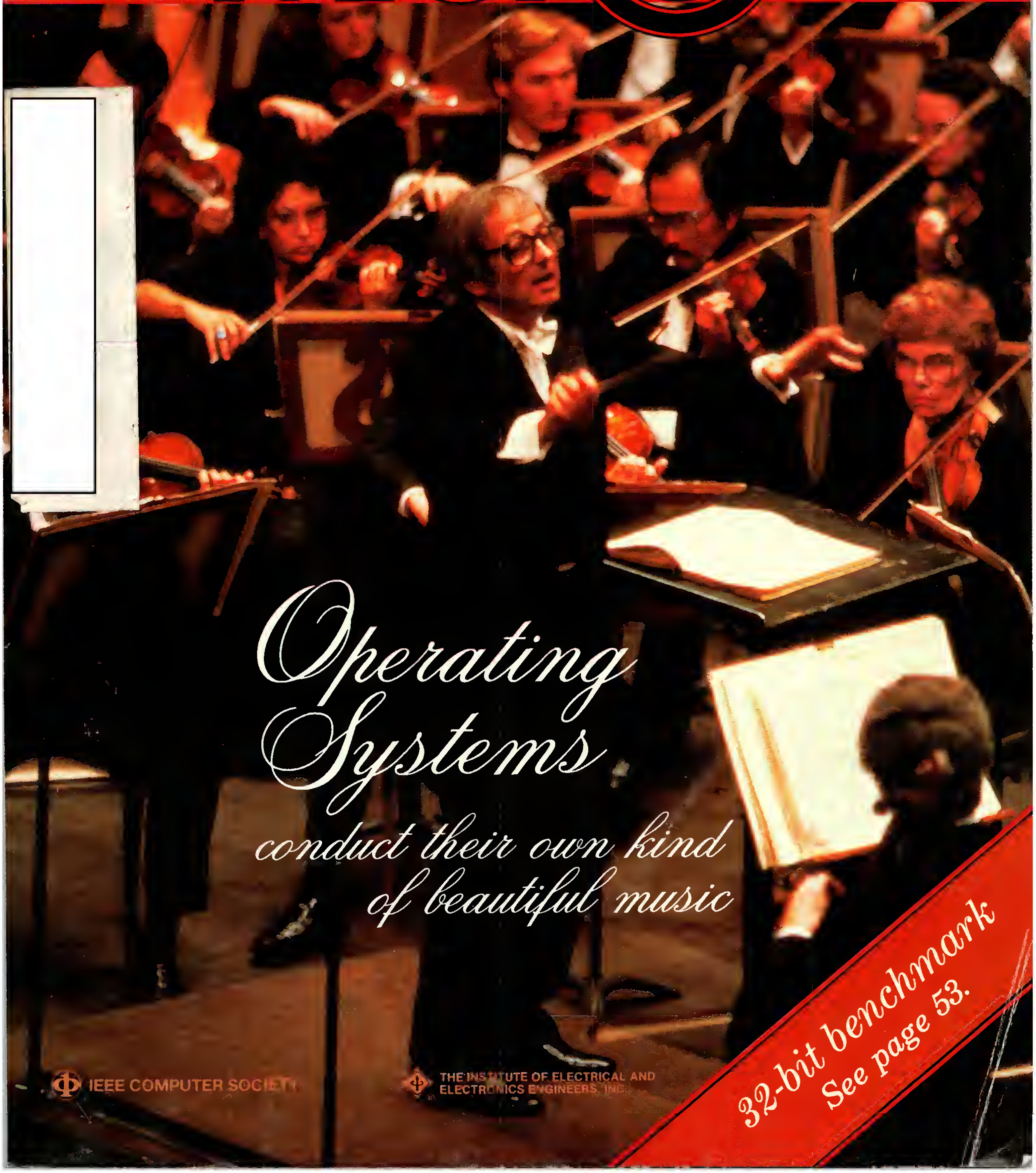
IEEE MICRO

AUGUST 1986


Operating Systems conduct their own kind of beautiful music
32-bit benchmark
See page 53.
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

ISMM

CALL FOR PAPERS

SOFTWARE AND HARDWARE APPLICATIONS OF MICROCOMPUTERS

February 4-6, 1987

Location

University Park Holiday Inn, Fort Collins, Colorado, U.S.A.

Sponsors

The International Society for Mini and Microcomputers—ISMM
Technical Committee on Computers

Scope

All aspects of microcomputers, their applications, software, interface, hardware, communications, etc., are of interest. Papers are invited which fall into, but are not limited to, the following subject categories:

- Adaptive systems
- Agriculture
- Algorithms for computer control
- Biomedical engineering and instrumentation
- Communication
- Computer vision
- Data base systems
- Digital control systems
- Distributed systems
- Expert systems
- Fault tolerant computers
- Finance
- Guidance, control and navigation
- Hardware architecture
- Image processing
- Management
- Networks
- Parallel processing
- Pattern recognition
- Robotics
- Software reliability
- Software testing
- Speech analysis and reconstruction
- VLSI systems in control and digital signal processing
- Other applications

Categories of Papers

Regular papers   Survey Papers   Tutorials   Short communications

International Program Committee

General Chairman: G.K.F. Lee; Program Chairman: F.F. Yassa

B. Johnson	Y. Fong	M.B. Zaremba	S.S. Udpa
D. Cannon	Y. Kim	M. Savic	P. Visuri
G.T. Catalano	C. Looney	M.A. Soderstrand	

Address

For submission of abstracts: Dr. Fathy F. Yassa, Corporate Research and Development Center, General Electric Company, P.O. Box 8, KWC510, Schenectady, New York 12301, U.S.A.

Abstracts

Three copies of a 200 word abstract and three copies of a 1000 word maximum extended summary of the paper should be received by Dr. F. Yassa prior to October 1, 1986. Both the abstract and the extended summary should show the authors' names and addresses for communication. Notification will be mailed by November 1, 1986. The photo-ready manuscripts are due at the beginning of the conference for publication in the proceedings. All papers will be reviewed for possible publication in the ISMM journals *The International Journal of Mini and Microcomputers* and *Microcomputer Applications*. ISMM holds the copyright for the publication of the papers. Requests for the organization of sessions may be directed to Dr. G.K.F. Lee.

'87

MIMI '87
FORT COLLINS

Please complete and return this form to: ISMM Canadian Secretariat, Box 25, Station G, Calgary, Alberta, Canada T3A 2G1.

Last Name .... First Name ....

Company .... Position ....

Address .... Tel. ....

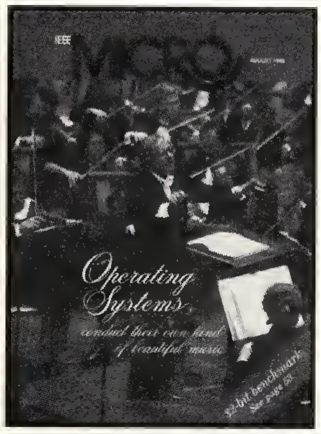
City .... Country .... Zip Code ....

Please send me information concerning:

☐ ISMM membership.
☐ The International Journal of Mini and Microcomputers.
☐ Microcomputer Applications Journal.
☐ Mini and Microcomputers and Their Applications, Austin, Texas, U.S.A., November 10-12, 1986.

IEEE MICRO

Volume 6 Number 4 (ISSN 0272-1732) August 1986


IEEE
MICRO
Operating Systems
conduct their own kind of beautiful music
32-bit benchmark See page 53

On the cover
Just as the conductor of an orchestra (in this case Andre Previn and associates) controls, schedules, and manages various instruments to create an outstanding musical offering so do operating systems control, schedule, and manage various resources to achieve an efficient system.

Photo courtesy Los Angeles Philharmonic Assn.

Cover: Jay Simpson and Larry Keiser
Article design/production: Alexander Torres
Typesetting: Fagen Graphics

Feature Articles

IEEE MICRO

Editor-in-Chief: James J. Farrell III, VLSI Technology Incorporated*

Associate Editor-in-Chief:
Joe Hootman, University of North Dakota

Editorial Board:
Shmuel Ben-Yaakov, Ben Gurion University of the Negev
George S. Carson, GSC Associates
Dante Del Corso, Politecnico di Torino, Italy
K.-E. Grosspietsch, GMD, Germany
David B. Gustavson, Stanford Linear Accelerator Center
David L. Hannum, AT&T Information Systems
Victor K. L. Huang, AT&T Information Systems
Barry W. Johnson, University of Virginia
David K. Kahaner, National Bureau of Standards
Kenji Kani, Nippon Electric Company
Henriecus Koeman, John Fluke Manufacturing Company
G. Jack Lipovski, University of Texas
Kenneth Majithia, IBM Corporation
Richard Mateosian
L. Robert Morris, Carleton University and DSPS Inc., Ottawa
Varish Panigrahi, Digital Equipment Corporation
Ken Sakamura, University of Tokyo
Richard H. Stern

Magazine Advisory Committee:
Dennis R. Allison (chair), Dharma P. Agrawal, Vishwani D. Agrawal, James J. Farrell III, Lansing Hatfield, Michael C. Mulder, David Pessel, True Seaborn, Bruce D. Shriver

Editor and Publisher: True Seaborn
Managing Editor: Marie English
Contributing Editor: Joe Schallan
Assistant to the Publisher: Patricia Paulsen

Advertising Director: Michael Koehler
Advertising Coordinator: Carole L. Porter
Membership/Circulation Manager: Christina Champion
Staff Writer: Torrey Byles
Art Director: Jay Simpson
Production Supervisor: David Gaines

*Submit six copies of all articles and special-issue proposals to James J. Farrell III, 10220 South 51st Street, Phoenix, AZ 85044; (602) 893-8574.

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society: IEEE Headquarters, 345 East 47th St., New York, NY 10017; IEEE Computer Society West Coast Office, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578. Annual subscription: $12.00 in addition to IEEE Computer Society or any other IEEE society member dues. Nonmember prices: available on request. Single-copy prices: members $7.50; nonmembers $15.00. This journal is also available in microfiche form.

Undelivered copies: Send to 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.

Postmaster: Send address changes to *IEEE Micro*, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854. Second class postage is paid at New York, NY, and at additional mailing offices.

From the Editor-in-Chief

Benchmarks!

When Thayne Cooper of Sperry-Univac called a few months ago proposing an article on microprocessor benchmarking, it provided me with a real flashback. It was over five years ago that two august members of your current *IEEE Micro* Editorial Board—Richard Mateosian and I—were present at the landmark *EDN* Carnegie Mellon benchmark meeting in the old Cahners building on Columbus Avenue in Boston. Of course, neither of us were associated with *IEEE Micro* at that time. We were among the participants. A lot of discussion took place during that meeting, and even now, years after the benchmark results were published in *EDN,* I still see references to it occasionally.

Thayne Cooper submitted the article, and it was reviewed as are all other articles. While it is Computer Society policy not to reveal the names of the referees of a specific manuscript, I can assure you that Mateosian and I were not the referees. The article appears in this issue on page 53. As always, your comments are invited.

The remainder of this issue I think you will find interesting also. Dick Stern continues his fascinating discussion on the legal aspects of integrated circuit design and copying; Bob Stewart gives us a humorous insight into his work on bus standards over the years as his farewell column; Victor Huang presents selected refereed articles on operating systems; and of course we include the other departments and features that you have come to expect. One absence this month is Dave Hannum, whose column is missing due do to labor difficulty. Dave will return in October.

Since the last issue I have attended the National Computer Conference. This is a key conference for several Computer Society committee meetings as well as the annual Editorial Board meeting for *IEEE Micro.* I regret to report that the poor attendance at NCC further worsened the current financial picture for the Computer Society. Several actions are being put in place to control expenditures and keep us solvent. The low turnout at NCC is a mystery to me. After seeing fairly large crowds at Electro and Mini-Micro in Boston, I expected a general resurgence in conference and convention attendance. Things are better this year than last year at this time, but we are not out of the woods yet.

This month's mailbag contained 31 comment cards. One comment that I receive fairly often is a request for more coverage on new products. This is very difficult for a magazine of our size and frequency. The biweekly trade magazines will each publish four or five issues before *IEEE Micro* comes out with the October issue. We will continue to publish highlights on new products that we feel are of interest to you, but we just do not have the space to cover all the products that deserve coverage.

Other comments were:

"...VME...very welcome." J.A.G., Fishkill, NY
"Too much VME in this journal." D.Q., Urbana, IL
"...(requests) an article on Sperry VLSI-1100." R.M., Houston, TX
"...very happy with *Micro*...publish monthly...." B.H.H., Albuquerque, NM
"...(like to see)...Novix NC4000 Forth chip." D.L.M., Wichita Falls, TX
"...(I liked) everything. More! More!" I.A.S., Cambridge, MA
"...more new products...." B.D.B., Houston, TX
"...more recruiting ads...." P.M.V., Cedar Rapids, IO
"...more multiport serial communications ICs...." S.L.J., Austin, TX
"...(I dislike)...too specialized designs...." P.A.B., Pisa, Italy

Please keep the cards and letters coming. Your requests are a factor in the *IEEE Micro* editorial calendar. In addition to this issue focusing on operating systems, the October issue on multiprocessing (which has an incredible 16 articles in review), and the December issue on digital signal processing, 1987 looks like a leading-edge year also. Watch for the TRON issue from Japan, a special issue from Europe, an issue devoted to new-generation microprocessors, and more.

A final note: After this issue, Kenji Kani of NEC, Japan, having completed two consecutive two-year terms as editor, leaves the Editorial Board of *IEEE Micro.* On behalf of the staff, volunteers, and readers, I would like to thank Dr. Kani for his hard work and contributions to *IEEE Micro*'s technical excellence—a job well done.

Jim Farrell

Jim Farrell

To the Editor

Design case study

To the Editor:

It might be of interest to readers to walk through a design case study in India, one of the developing nations of the world. At the Indian Telephone Industries, Bangalore, where I worked as assistant executive engineer from 1975-1978, the R&D unit was judged to be the best in the country in the "public" sector, i.e., run by the government or a quasi-governmental organization. The then chief engineer, who was eventually promoted to general manager, was responsible for enlarging the R&D wing from a handful of people to one with more than 100 engineers. He was awarded a prize by the Indian government for his achievement.

The Indian Telephone Industries and the Indian Post and Telegraphs Ministry operated much like the Bell system before divestiture. ITI was a captive unit for P and T, and all its products, except a few, went to that organization.

The problems with a country like India are manifold when it comes to designing and manufacturing a sophisticated electronics product—in this case a microprocessor-based audio test set for making response and return-loss measurements on telephone lines. The ATS (as it was called) was also slated to measure parameters on audio systems like amplifiers, equalizers, and filters.

Although India is now the 10th largest industrial power in the world, there is still a lack of an adequate technical infrastructure; most of our integrated circuits are imported from the USA, and substantial red-tapism exists for the procedure to import a component, because foreign exchange is scarce and competition for its use is keen. Also, the information bottleneck that exists between India and the western world made the making of a sophisticated instrument a daunting task.

The decision to incorporate a microprocessor in the ATS was not made initially; it was incorporated into the design at a later stage of iteration. Anyway, there was a substantial digital content in the ATS.

The instrument was to have a basic accuracy of 0.1 dB, with a built-in CRT display for the measured parameters; a swept-frequency oscillator between 20 Hz and 20 kHz with presettable end-points; a digital frequency meter for the marker frequency; a digital dB meter for the send and receive signals; a digital storage system for slow traces and leisurely viewing; and a provision to measure return-loss, frequency-response, pre-emphasis and de-emphasis curves, etc.

We had, as a tentative model, a Siemens 200D-type instrument—which, however, has a much smaller bandwidth and fewer features, and a Halcyon audio test set. We followed the Siemens mechanical construction in a modified form, and, as far as the electronics was concerned, we went in for a higher level of integration, chiefly because there was only one full-fledged engineer (myself) working on it, and so, to shorten the design cycle, the "make or buy" decision was often weighted in favor of the latter. We used circuit modules like high-voltage op amps for the CRT deflection amplifiers whenever an advantage was perceived over using integrated circuits. We had a mind to use the Intel 8085, and we used a FIFO memory for the digital storage mechanism. For the CRT we used a Philips high-accuracy instrumentation CRT and imported it from Holland. From the raising of the purchase requisition to the arrival of the CRT was a good 18 months.

The way we went about ordering components was this: We would look at the trade magazines like *Electronics* and *Electronic Design* and choose suitable components like ICs. Then we would order them. By the time we received them, an average time of eight months, there was already a better product on the market. We were doomed to obsolescence. I think India is at the stage where Japan was 30 years ago; nobody thought that the Japanese could make an acceptable product.

The R&D laboratory worked on the matrix method, with workers reporting to the project leader for the "what" of design, and to their departmental boss for the "how" of implementation. Thus the senior technical assistant in the Mechanical Shop took my drawings to make the cabinet and mechanical fittings but worked under the guidance of the senior engineer in the Mechanical Shop. I had another STA working under me for the electronics, and I reported to the senior engineer in the Instruments Group.

The Computer-Aided Design Group designed some of our resistor arrays and delivered them to us on a ceramic substrate; they designed some of the board layout too. India is always interesting in the worship(sic) of Janus, the Roman god who gave his name to the month of January: We look forward to rockets but also keep the bullock cart in view. And thus it was that some of the artwork on the glass-epoxy boards were done with manually wielded paintbrushes. Remember the American engineer who spent a year in the Japanese factory? Lunch boxes, with "fish of the day," came in a manpowered three-wheel vehicle.

It was a heady time for us; we paid attention to ergonomics; to the shape of the CRT hood for wide-angle viewing; to choosing green LEDs for the display for soothing tired eyes; to the position of the control knobs for easy accessibility—concepts which have become blase at HP and Tektronix perhaps but are still relatively new in India.

In the final stage of iteration we kept in mind the words of Dave Methvin of Computer Automation: When designing a TV set, throw out things, and keep on throwing, until you hurt your sound or picture. Then stop. When making a case for a computer, don't make it so strong that a man can stand on it. Our computers are not meant to be stood upon!

If a Blake-Moulton managerial grid were to be drawn for our chief engineer, I think his vector would lean heavily toward the "people" axis, which made working in his lab a pleasure and not a chore.

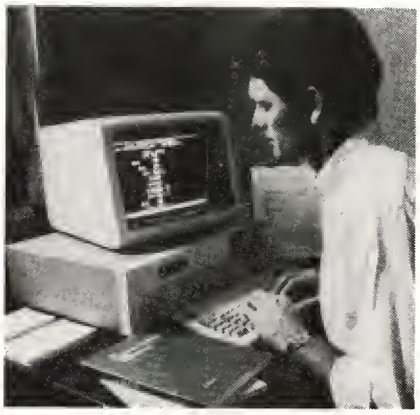
We had weekly review meetings, where we brawled over specifications—and we had occasional parallels to National's case for cutting down on chip burn-in times.

We had a bit of serendipity, too, when a local firm suddenly started making precision resistors and saved us the delay and foreign exchange involved in importing.

I think if I had to do it all over again, I would rather have a team working on it. I missed the synergistic effect of working in a team. I had to study everything from noise problems to CRT power supplies to mechanical mounting of the CRT. When I left in 1978, the

continued on page 79

Announcing new PC SIMSCRIPT II.5 . . . with animation


SIMSCRIPT II.5 with animation now on personal computers

free trial--see how SIMSCRIPT II.5 helps you build a realistic model

the complete SIMSCRIPT II.5 on a PC

SIMSCRIPT II.5 for personal computers is the same popular simulation language that is now widely used on mainframes.

You can now build realistic models of military, manufacturing, communications, logistics, transportation or other systems on your PC.

PC SIMSCRIPT includes a new programming environment that makes it easy for you to develop, verify, modify, and enhance simulation models on a personal computer.

natural method of modelling

When building a model in SIMSCRIPT, you describe the simulated system as consisting of certain types of entities: perhaps workers, machines and jobs in a simulated factory; or flights and airports in a simulated air transport system; or jobs, processors, channels, and I/O devices in a simulated computer system.

For each type of entity you give names to the attributes that characterize it.

You also name the sets an entity type may belong to, and the sets it may own.

Since your model is English-like, with names that you choose, it reads like a description of the simulated system. The model can be read and verified by non-programmers who understand the system under study.

This makes your model development, validation and evolutionary changes much easier.

large models on your PC

Your model and data are not limited by the size of the PC. SIMSCRIPT is the only simulation tool that automatically makes use of the hard disk as a memory extension.

reduced cost

SIMSCRIPT II.5® is a well established, standardized, and widely used language with proven software support.

Experience has shown that SIMSCRIPT II.5 reduces simulation programming time and cost severalfold compared to other simulation techniques.

animated and graphical results

With PC SIMSCRIPT II.5® you build models that can show an animated picture of the system under study. Observing the simulation improves understanding of the system and builds confidence in the model.

Because you see the operation of the simulated system and can easily try alternatives, the time and cost of system analysis are sharply reduced.

computers with SIMSCRIPT II.5

1. IBM Personal Computer AT, XT, PC or compatible, with a hard disk.

2. Most Mainframe computer types including IBM, CDC, VAX, Univac, Prime, Gould, Data General and Honeywell.

SIMSCRIPT II.5 and PC SIMSCRIPT II.5 are registered trademarks and service marks of CACI, INC.-FEDERAL

free trial

The free trial package contains everything you need to try SIMSCRIPT II.5 on your own computer.

We send you PC or Mainframe SIMSCRIPT II.5, installation instructions, sample models, and a complete set of documentation. You can build your own model or modify one of ours. No cost or obligation.

special offer free training

For a limited time we will also include free training. **Space is limited so act now to avoid disappointment.**

Call Rick Crawford at (619) 457-9681 to reserve your place.

free trial-learn the reasons for the broad and growing popularity of SIMSCRIPT II.5—no cost or obligation

special offer-return the coupon today and we will include one free course enrollment worth $850

Name ____________________

Organization ____________________

Address ____________________

City __________ State __________ Zip __________

Telephone ____________________

Computer __________ Operating System __________

Return to: IEEE MICRO

CACI

3344 North Torrey Pines Court
La Jolla, California 92037

Or, better yet,
call Rick Crawford at (619) 457-9681

Guest Editors' Introduction

Operating Systems

Victor K. L. Huang and Priscilla M. Lu
AT&T Information Systems

As we approach the 21st century and enter the Information Age, we see the uniquely qualified electronics industry playing a vital role in realizing the new era by providing state-of-the-art technology through microelectronics, telecommunications, computer networks, and software. The advent of 32-bit microprocessors that offer the processing power of minis at a fraction of the cost opens up new applications for the ubiquitous microchips in some areas previously thought impossible for computing. This is especially so in the automation of services and business industries.

Concurrent with these microtechnological advances are applications in the office, factory, and home, in entertainment and business, that have spawned new areas of opportunities. Office automation, factory automation, and home and business computers (PCs, desktops) make "computing for the masses" a reality. No longer is computing confined to big mainframes with traditional centralized processing and software and operating systems geared toward enhancing the performance of number-crunching operations.

With microelectronics we use microprocessors in workstations and file servers—be it in business, office, or scientific endeavors—which enhance the productivity and information management of the end user. In more-automated environments such as industrial automation, we find microprocessors embedded in controllers and monitors—very often controlling real-time, multitasking operations. Process control in factories, security and alarm operations in nuclear reactors, and patient monitoring in hospitals are some vivid instances of such applications.

Many of these controllers link hierarchically at interfaces to a central processing unit, or a network of processors, which provides for the management of more information or for more information-processing capabilities. The support of these complex multiple-node architectures requires advancement and innovation in the operating systems. Operating system architectures have evolved over the last decade to fully exploit multiple-processor distributed systems. Operating systems, in all flavors, are being pursued to provide effective end-user interfaces, while complying with constraints of code compatibility, portability, and hardware transparency.

In planning for this special issue, we focused on operating system architectures and analysis that exploit the rapid advances of microelectronics technologies. We examine a real-time operating system that is designed and implemented specifically for microprocessors embedded in the industrial environment; the architecture of a distributed operating system, the V-system; and finally an implementation and analysis of a network file server.

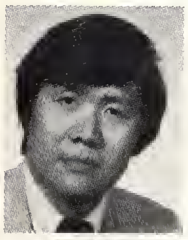
The first article by James Ready, of Hunter & Ready, Inc., discusses the Versatile Real-Time Executive (VRTX) family of software components. Designed specifically for embedded microprocessor applications in industrial environments, VRTX provides necessary real-time and concurrency control for software that must meet the response-time requirements of industrial systems.

Implementation of distributed file servers seems to be a current topic in the advances of operating systems technology today. Therefore, it is timely that the next article addresses this issue. Bill Jackson of EDS discusses implementation issues in the architecture of a file server having the major requirements of availability and reliability.

In the next article Eric Berglund of Stanford University describes the V-System architecture. It is a distributed hardware configuration that is available to client/server systems. The primarily message-based system supports a functionally partitioned operating system architecture wherein different processors assume responsibility for different tasks. The heart of the system is the distributed kernel (V-kernel), which provides inexpensive processes and fast interprocess communications. Servers use concurrency and message passing to run at the user level but also provide operating system functions. Berglund describes the design of the servers, along with their use of the V-kernel primitives and the interactions between them.

We feel these articles address issues that provide some insights into the advances made in operating systems and their environments and thus should contribute to the overall advancement of technology in this information age.

We acknowledge the many reviewers for their timely comments and appreciate the time they took in the review process. We also fervently hope that this exciting field of operating systems technology returns as a special theme to *IEEE Micro* soon.



Victor K. L. Huang is supervisor of the VLSI Processor Development II Group in AT&T Information Systems and holds responsibilities for the design and development of 32-bit VLSI microprocessors for the WE32100 microprocessor product line. He has supervised new and evolving directions and specifications in VLSI systems architecture and associated components for Bell Telephone Laboratories. He was the principal architect and designer for the single-chip, 4-bit WE4000 microcomputer. His major field of interest is in microelectronics and VLSI architectures and systems, including processor architecture and design, computer modeling and performance evaluation.

Huang holds PhD and MS degrees from the University of Virginia and the BS degree from Virginia Military Institute, all in electrical engineering. He also holds four patents in digital logic and microprocessor design and is author and coauthor of 12 publications.

He is vice president, administration of the IEEE Industrial Electronics Society and associate editor of *IEEE Micro*. He has been very active in many IEEE conferences and seminars.

Priscilla M. Lu is department head of Microsystems Engineering where she is responsible for product line architecture, requirements, and delivery plans, including quality certification, competitive analysis, and performance evaluation. She also holds responsibility for Unix development for the product line and exploratory work in VLSI architecture. She previously supervised the Microprocessor Systems Architecture Group, which was involved in VLSI chip set architecture and design. As an engineer she worked on the development of the 3BNET, Datakit, and Ethernet network designs. Her interests lie in VLSI computer architectures and design, local area network and telecommunication systems, and software system design (operating systems and languages).

Lu holds a PhD in EE&CS from Northwestern University and an MS and BS in computer science and mathematics from the University of Wisconsin, Madison. She has authored over 25 publications and holds two patents.

The guest editors may be contacted at AT&T Information Systems, 2M-305 Crawfords Corner Road, Holmdel, NJ 07733.

VRTX: A Real-Time Operating System for Embedded Microprocessor Applications

Programmers can speed the development of complex software for powerful 32-bit microprocessors by designing with reusable software components.

James F. Ready
Hunter & Ready, Incorporated

The advent of the microprocessor has opened up hundreds or even thousands of product opportunities that simply did not exist earlier. The latest generation of 32-bit microprocessors has brought another quantum leap in the power of the devices themselves, and their application in industry promises to produce another explosion in advanced products based on microprocessor technology.

In recent years, however, a bottleneck has developed that threatens to retard the progress of new microprocessors into industrial applications. The development and maintenance of software has become a critical stumbling block for these new machines. They are as powerful as the minicomputers or even large computers of a few years ago, and they require software that is equally complex.

The obvious solution to the software problem is to use the techniques that have worked so well in reducing the costs of hardware. The basic reason for the decline in hardware costs is the use of standard large-scale integrated circuits. These standard components are hooked together by board designers who never need to understand their inner workings.

In September 1981 Hunter & Ready, Inc., introduced the first of the the VRTX (Versatile Real-Time Executive) family of software components designed and implemented specifically for microprocessors used in process control, instrumentation, communications, and intelligent computer peripherals. These software components can be used exactly like LSI hardware components; that is, they can be used in any hardware environment with absolutely no modification to the component. Software for microprocessors used in industrial applications can now be constructed just like hardware. The user no longer pays the development and maintenance costs of these software components.

By 1986 over 1000 applications worldwide had been designed using the VRTX software components. In a very real sense, we feel the dream of reusable software has become a practical, everyday reality.

Software for industrial environments is characterized by special requirements—a description of which opens the discussion below. What follows is a technical discussion of the way we designed a true, hardware-independent software component.

Special needs of software used in embedded applications

Computers can be divided into two classes, general-purpose and embedded. A personal computer is a good example of a general-purpose computer. It is visible to the user as a computer; it is reprogrammable by the user; it has standard computer peripherals such as a keyboard, display, and disks; and its software has a general-purpose human interface.

An embedded computer presents a much different picture. The main role of a computer in an embedded system is to serve as an information processing component within a larger engineering system. Typical examples are factory instrumentation and control equipment, file and print servers for office automation, telecommunications switches, aircraft avionics, and industrial robots. In each of these cases the user need not know that a computer exists inside the equipment; the computer is typically not reprogrammable by the user; it has a variety of unusual I/O devices (sensors, actuators, and the like); and the interface to the user, if there is one, is application specific.

Software used in embedded computer systems is similar in many respects to software used in general-purpose computers (for example, the C language is commonly used in both environments). However, there are several unique characteristics of embedded computers that impose some additional requirements on the software used in these environments.

Real-time and concurrency requirements. An embedded computer is connected directly to the physical equipment and is dedicated to controlling that equipment. Consequent-

Table 1.
VRTX tasking system calls.

System call	Definition	Action
SC_TCREATE	Task create	Creates a new task, assigning it a priority and an ID.
SC_TDELETE	Task delete	Deletes a task.
SC_TSUSPEND	Task suspend	Suspends execution of a task.
SC_TRESUME	Task resume	Resumes execution of a suspended task.
SC_TPRIORITY	Task priority change	Changes the priority of a task.
SC_TINQUIRY	Task inquiry	Returns the status and priority of a task.
SC_LOCK	Disable task rescheduling	Makes the running task unpreemptible.
SC_UNLOCK	Enable task rescheduling	Makes the running task preemptible.
SC_TSLICE	Enable time-sliced scheduling	Turns time-slicing on or off.

ly, the system must meet response requirements that are mandated by the equipment itself, rather than those being dictated by the computer. The requirement to meet externally imposed deadlines is at the heart of what we term a real-time system. The definition of a real-time system is one that has to respond to externally generated events within a specified and finite interval.

If the embedded computer produces the right results late, it might as well have produced the wrong results. Closing a valve after gallons of radioactive water have spilled out has much the same results as if the valve were never closed at all.

Most embedded systems have a real-time requirement. Consequently, the software used in such a system must be designed to meet these response time requirements.

Another characteristic of embedded computer systems is that many activities proceed in parallel within the computer. For example, some parameters must be sampled and controlled at a very quick rate, whereas other parameters of the process need only be sampled once a second. Logically, those two operations proceed in parallel, while of course the CPU executes them in an interleaved fashion. Similarly, controlling several I/O devices at once usually results in some parts of the control software waiting for the devices to complete an operation while other devices, having finished their operations, need to be serviced.

Whether recognized or not, this parallelism, or concurrency, adds a major complication to the software, and failure to recognize this important fact can result in system failure. To avoid this kind of problem, programmers need to be able to guarantee mutually exclusive access to shared resources, signal one task by another task or from an interrupt handler, and send messages from one task or an interrupt handler to another task.

VRTX supplies the multitasking facilities necessary to solve the real-time performance requirements found in embedded systems. In this approach the CPU is multiplexed among several modules of software, called tasks, that compete for the CPU. Each task is given a priority that reflects its relative urgency in controlling the CPU. When an important event occurs, the CPU is switched to the task that services the event. By carefully setting the priorities of the tasks in the system, the designer can allocate the CPU time appropriately to meet any real-time deadlines (providing, of course, that the CPU has enough power to handle the load). For background on the multitasking approach to real-time design and implementation, see both Wirth and MacLaren for excellent discussions on the subject. [1,2]

VRTX also provides mutual exclusion, signaling, and intertask communication services, along with the basic multitasking functions. Together, the programmer has the basis for a reliable design that can handle both real-time and concurrency requirements. We describe the details of these as well as some other useful system services in the following sections.

Task services for real-time and concurrency control. VRTX provides system calls for creating and deleting tasks, suspending and resuming their execution, and so on. In addition the system automatically takes care of task scheduling. Although tasks may logically proceed in parallel, the CPU is physically capable of running only one task at a time; therefore, the VRTX scheduler interleaves the execution of tasks on the CPU.

To ensure the most rapid response possible to real-time events, VRTX employs a preemptive priority-bases scheduling technique. Each task is assigned a priority when it is created; the priority value reflects the task's relative urgency. When more than one task is ready to run, VRTX always selects the highest priority task. Further, whenever it executes a system call that may ready a task, VRTX reschedules; that is, after performing the requested service but before returning to the calling task, VRTX determines whether execution of the call has made a higher priority task ready. If it has, VRTX immediately runs the higher priority task, and the status of the lower priority task is changed from executing to ready. In this way an urgent task begins to run the instant it becomes eligible. Table 1 lists the VRTX tasking system calls.

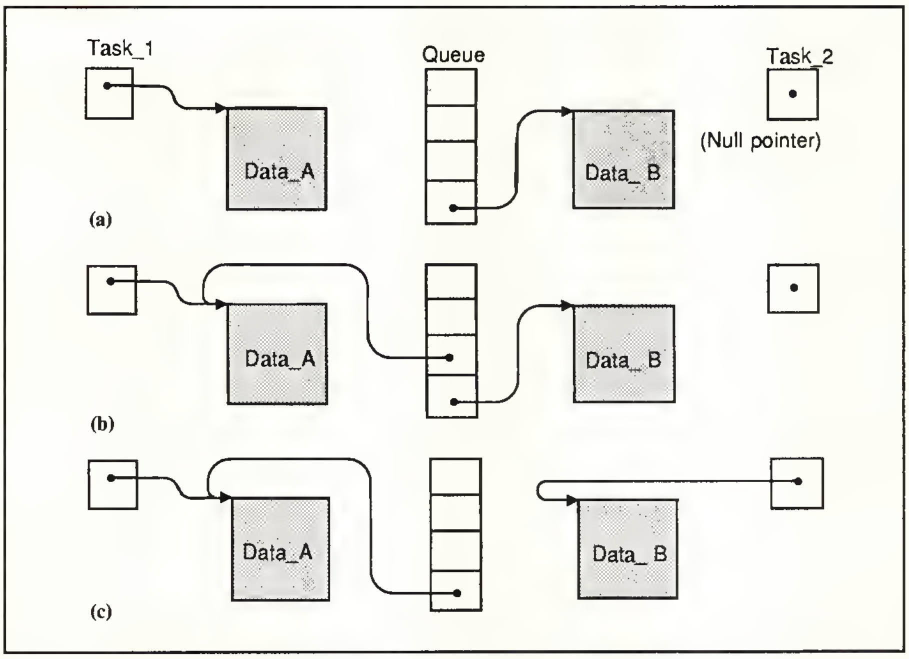
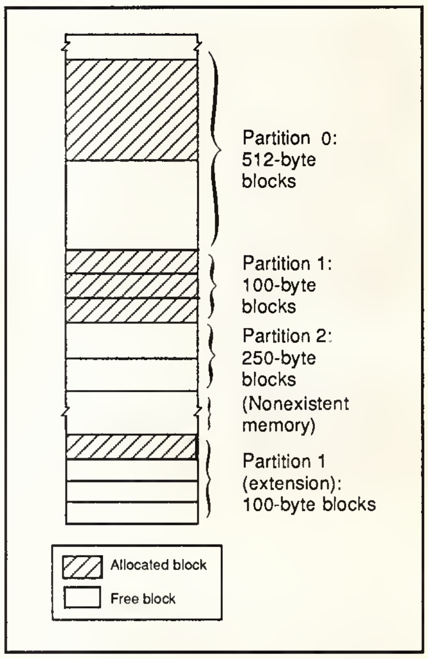


Figure 1. Basic queue operations. Initial situation: A four-element queue contains one message. (Both tasks have pointers to the queue, which are not shown in clarity) (a); after Task_1 sends a message (pointer to Data_A) to the queue (b); after Task_2 receives a message (pointer to Data_B) from the queue (c).



Partition 0: 512-byte blocks
Partition 1: 100-byte blocks
Partition 2: 250-byte blocks
(Nonexistent memory)
Partition 1 (extension): 100-byte blocks
Allocated block
Free block

Figure 2. Partitioned memory pool.

Table 2.
Intertask synchronization and communication system calls.

System call	Definition	Action
SC_POST	Post message	Places a message in a mailbox. If the mailbox is full, notifies the caller.
SC_PEND	Pend for message	Obtains a message from a mailbox. If the mailbox is empty, the task is suspended until a message arrives or a specified time interval passes.
SC_ACCEPT	Accept message	Obtains a message from a mailbox. If the mailbox is empty, notifies the caller.
SC_QCREATE	Create message queue	Creates a new queue.
SC_QPOST	Post message queue	Adds a message to the end of a queue; notifies the caller if the queue is full.
SC_QPEND	Pend for message from queue	Obtains the message at the front of a queue. If the queue is empty, the task is suspended until either a message arrives or a specified interval passes.
SC_QACCEPT	Accept message from queue	Obtains the message from the front of a queue; notifies the task if the queue is empty.
SC_QINQUIRY	Inquire as to queue status	Returns the count of messages currently in the queue and the status of the queue.

Table 3.
Memory allocation system calls.

System call	Definition	Action
SC_GBLOCK	Get memory block	Obtains a block of free memory from a partition.
SC_RBLOCK	Release memory block	Returns a block of memory to a partition so it can be reused.
SC_PCREATE	Create memory partition	Creates a partition.
SC_PEXTEND	Extend memory partition	Extends a partition by adding more blocks to it.

Intertask mutual exclusion, signaling, and communication services. In VRTX tasks use mailboxes and queues to signal events, achieve mutual exclusion, and communicate between themselves. A VRTX mailbox is functionally equivalent to an event signal. The SC_PEND call allows a task to wait for a signal. The SC_POST call allows a task (or an interrupt service routine) to signal an event. Using mailboxes to implement this method of event signaling has a number of advantages over other commonly used signaling mechanisms. VRTX events can pass data along with the event, detect and report event overruns to the signaler, and signal events directly to a waiting task from an interrupt service routine.

A message is simply any pointer-sized quantity. The message may contain data, or it may be a pointer to data located elsewhere. By using pointers for messages, any type of information can be passed between tasks. Both mailboxes and queues decouple the execution of communication tasks by storing messages that have been sent but not yet received and by providing a place for tasks to wait for messages that have not yet been sent.

A queue (see Figure 1) can hold multiple messages. (The number is specified when the queue is created). This buffering facility is very useful when a sending task (or tasks) can temporarily produce messages faster than a receiving task (or tasks) can consume them. The simpler single-message mailbox, on the other hand, has the advantage of speed: Operations on it are faster than queue operations, making it the appropriate choice when message buffering is not needed.

Both mailboxes and queues are many-to-many communication channels: Any number of tasks can send and receive messages to and from the same mailbox or queue. When multiple tasks are waiting for a message to arrive at a mailbox or queue, the highest priority task actually gets the message. This form of message scheduling is essential for recognizing and handling high-priority events.

The VRTX system calls for communication are listed in Table 2. These calls synchronize tasks, as well as communicate between them. Intertask synchronization can be thought of as a special case of communication in which the content of a message is immaterial; it is a task's possession of a message that counts. A VRTX mailbox is functionally equivalent to the classic task synchronization facility, the binary semaphore. The SC_PEND call is identical to the P (or Wait) semaphore operation, and the SC_POST call corresponds to V (or Signal). Similarly, a VRTX queue is functionally equivalent to a buffering general, integer, or counting semaphore: SC_QPEND and SC_QPOST correspond to P/Signal and V/Wait respectively.

Memory services. Often the memory requirements of a task are not constant throughout its lifetime but vary depending on data it encounters or processing phases it passes through. For example, a task can both add an element to a linked list whenever it receives one kind of message and remove an element whenever it receives another kind of message. Substantial storage economies can be realized in cases like this by dynamically allocating and releasing list element storage at runtime. (Storage released by one task can be allocated to another.) VRTX supports dynamic allocation by maintaining a free pool of memory blocks; tasks can allocate and release blocks as they run.

VRTX's approach to dynamic memory allocation is based on the needs of real-time multitasking applications. These needs boil down to speed and, most importantly, predictability. In particular, it is not acceptable for a task's allocation request to be honored immediately most of the time or to be delayed occasionally due to the temporary unavailability of a block of the requested size. All variable-block allocation schemes can, under certain conditions, produce unpredictable response times; therefore VRTX allocates and releases storage in fixed-size blocks.

To obtain reasonable space efficiency, we can subdivide the free pool dynamically into partitions (see Figure 2), each of which may contain blocks of a different size. (There are no constraints on block sizes). By allocating from the partition with the block size closest to the actual amount of memory it needs, a task can minimize wasted memory. At the same time the fixed-block size of each partition enables VRTX to manage the pool with minimal overhead and no external fragmentation. Partitioning also permits the pool to span discontiguous address ranges that are commonly encountered in microprocessor systems (see Figure 2). See Table 3 for memory services system calls.

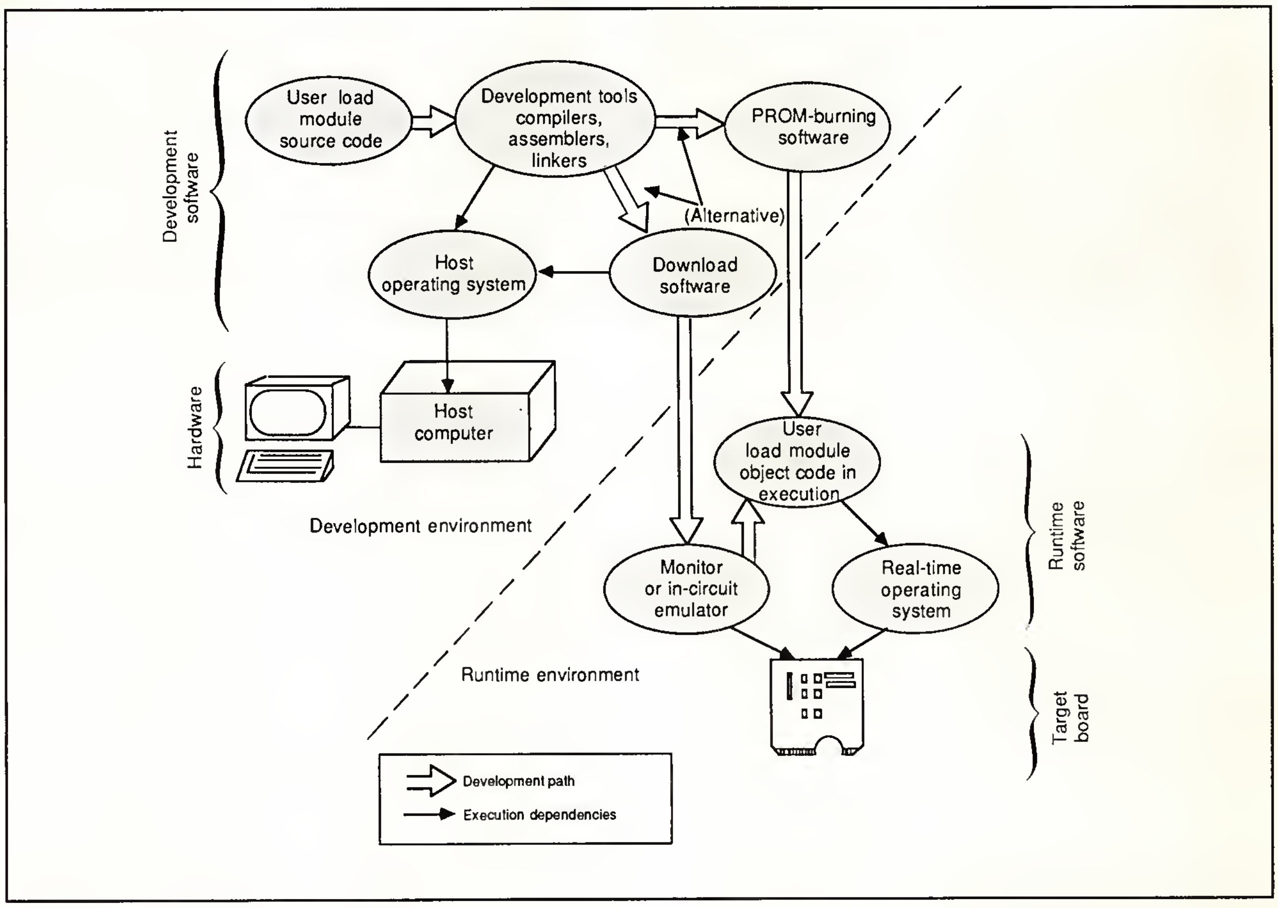


Figure 3. Cross-development path.

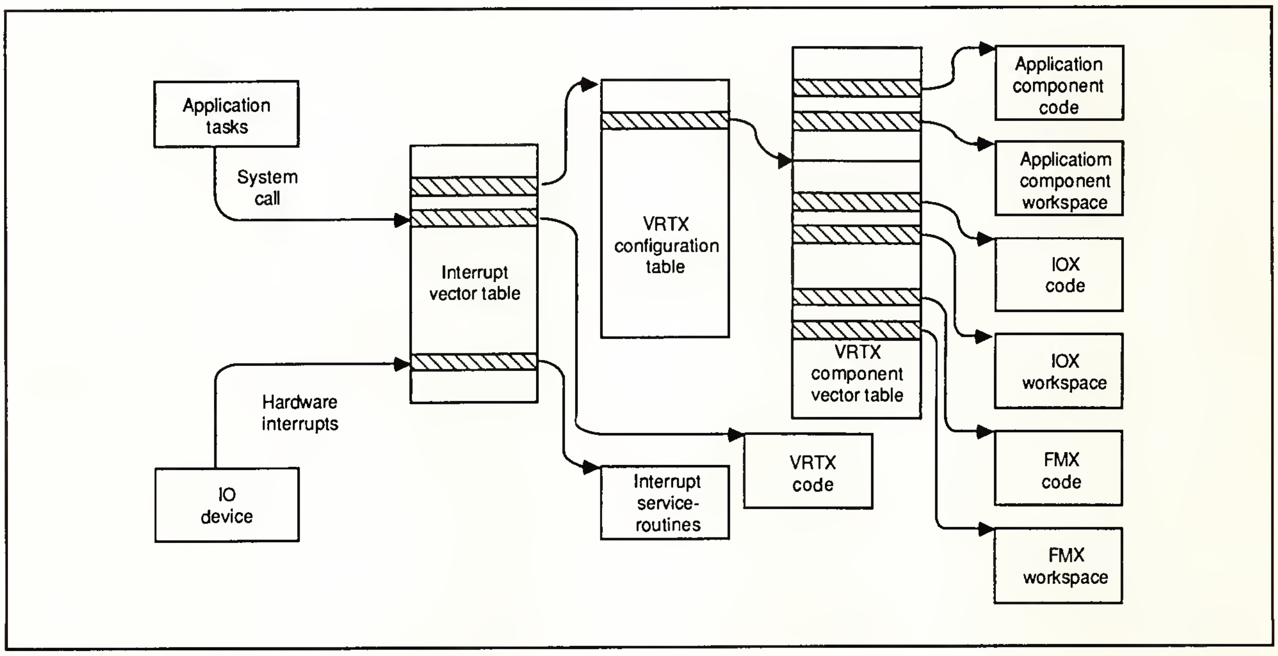


Figure 4. Software component interface.

Designing for development and target system independence

Figure 3 illustrates the typical development environment for an embedded computer system. There are a number of important features to note. First, the host and the embedded computers are different. The embedded computer typically is either a custom design using a standard microprocessor or a single-board computer based on one of a number of popular buses such as Multibus or VME or a complete system such as an IBM PC. The target microprocessor may be one of a number of popular chips; the most widely used are the Intel 8086 and the Motorola M68000 families.

A number of languages are currently used for programming embedded systems. These include C, Pascal, PL/M, and assembly. A further complication is that there are many implementations of each of these languages, each implementation of which may be unique in its internal structure.

In this environment we set the goal to build software components that were independent of the microprocessor selected (8086 or 68000), independent of the particular implementation (a board-level implementation of an 8086 with particular peripheral chips), independent of the particular high-level language and other tools used (C, Pascal, or PL/M, and specific linker formats), and independent of the particular development environment used (VAX VMS, Sun Unix, or an IBM PC with PC-DOS). A true software component is independent of all these hardware and software restrictions. Next we explain how we attained each of these goals.

Linking and locating independence. By using program-counter-relative and base-relative addressing, VRTX becomes position independent in memory. That is, it can be located anywhere in the microprocessor's address space without modification. Thus, the component need not depend on a particular linker or locator to assign internal addresses to fixed memory locations.

If there are other software components in the system, either supplied by us, by other suppliers, or by the programmer, VRTX can use a mechanism that allows these components to communicate with each other without needing to know any other component's location in memory. It works in the following manner.

When VRTX receives control as a result of the invocation of a system call, a processor register contains a two-part function code. As might be expected, one part of the code identifies the operation that is to be performed. The other part of the code, however, identifies the software component that is to execute the operation. This component may be VRTX itself, another Hunter & Ready component such as IOX (Input/Output Executive) or FMX (File Management Executive), or an application-supplied component, as illustrated in Figure 4. Up to 255 software components can coexist in a given application. If an operation is directed to a component other than itself, VRTX consults the user-supplied Component Vector Table to find the address of the component. VRTX passes control and parameters to the component according to an efficient and well-defined protocol. Using the same convention, software components can call each other. IOX, for example, uses VRTX's intertask communication services extensively.

System generation. Traditionally, the building of a system, or system generation, is done on a specific host computer. Since VRTX may not assume any particular host environment, we required another approach to system configuration. The approach taken was to delay the configuration of the system until the execution of VRTX itself on the target computer.

Before VRTX can run, it must know a few things about the environment in which it is installed. For example, it must know the starting address and extent of its RAM workspace (where it keeps local variables, the free pool, and so on). VRTX must also be aware of application-specified options, for example, the maximum number of tasks, and the addresses of hook procedures, if any. Both types of information are communicated to VRTX via entries in the user-coded VRTX configuration table. This simple table (it consists of about a dozen entries) can be located anywhere in memory.

As shown in Figure 5, VRTX predefines one entry in the CPU's vector table (called the Exception Vector Table in the 68000 family and the Interrupt Vector Table in the iAPX86 family) to contain the address of the configuration table. When VRTX is told to initialize itself (discussed in the next section), it locates the configuration table by reading this vector. A second user-defined vector points to VRTX's entry point (located eight bytes from its base address); system calls use this vector to invoke VRTX functions. The location of both entries in the vector table can be specified by the user. All other vectors are available for application use; typically, several of them point to the entry points of interrupt handlers.

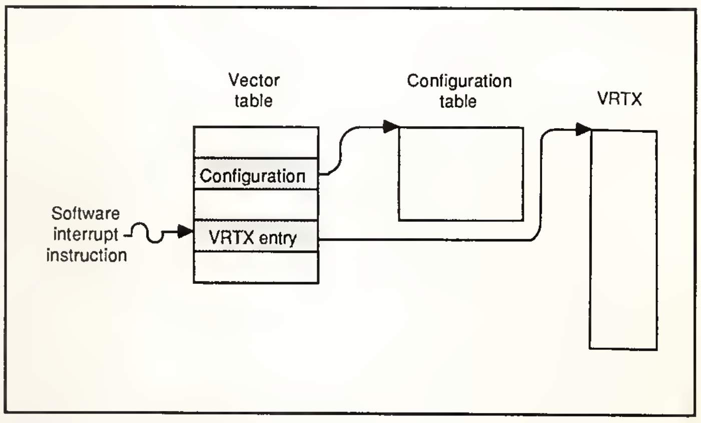


Figure 5. Basic VRTX links.

Table 4.
Initialization system calls.

System call	Definition	Action
VRTX_INIT	Initialize VRTX	Performs operations specific to initialization, such as setting up VRTX workspace, variables, task control blocks, and stacks.
VRTX_GO	Start application execution	Begins multitasking by executing the highest priority task created in the user's initialization code.

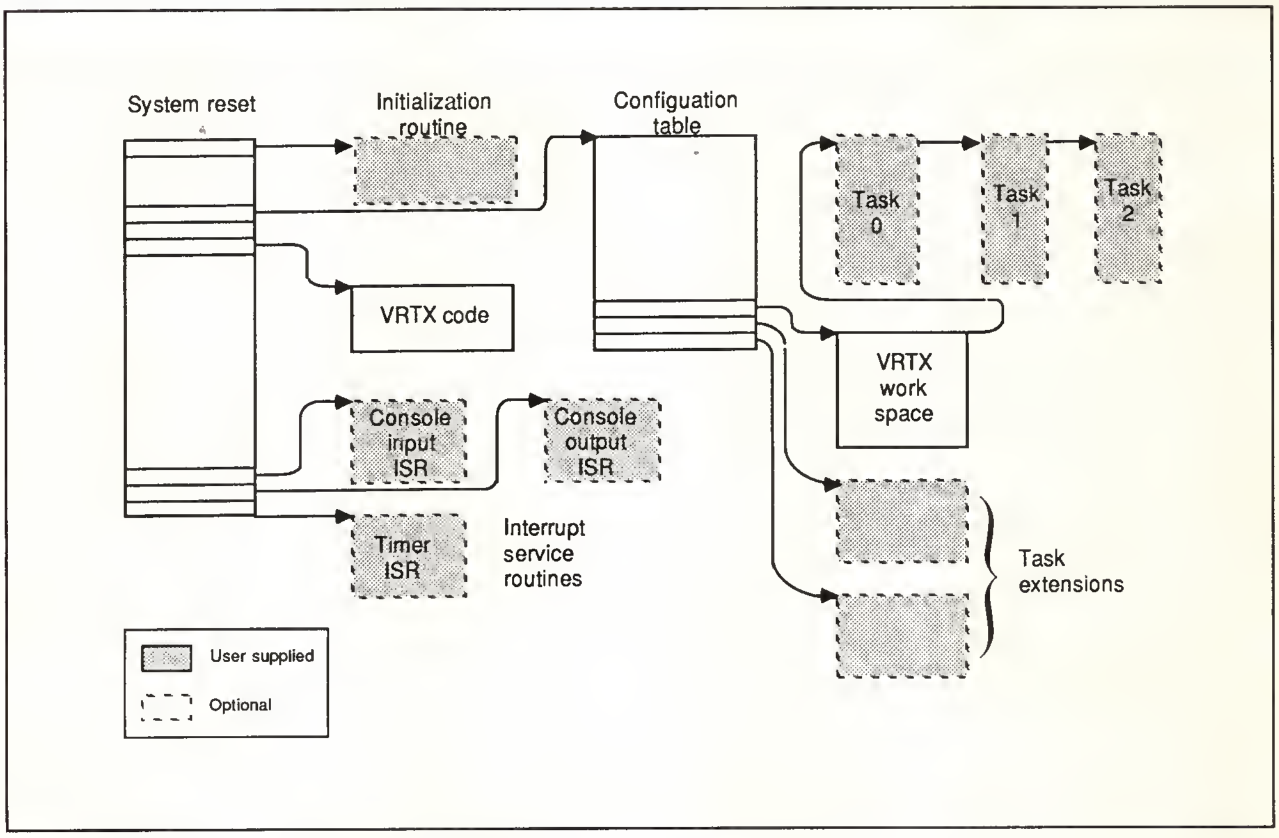


Figure 6. VRTX system following initialization.

Table 5.
Real-time clock system calls.

System call	Definition	Action
SC_GTIME	Get time	Obtains the current value of the clock counter.
SC_STIME	Set time	Sets the clock counter to a new value.
SC_TDELAY	Task delay	Suspends execution of the task for a specified number of clock ticks.

Table 6.
Character I/O system calls.

System call	Definition	Action
UI_RXCHR	Post received character from interrupt	Transfers a character obtained from the device to the VRTX buffer without causing rescheduling.
UI_TXRDY	Post transmit ready from interrupt	Obtains a character from the VRTX character buffer so it can be transmitted to the device; does not cause rescheduling.
SC_GETC	Get character	Obtains the next character from the device buffer; if the buffer is empty, the task is suspended until a character arrives.
SC_PUTC	Put character	Transmits a character to the device buffer; if the buffer is full, the task is suspended until space for the character becomes available.
SC_WAITC	Wait for special character	Suspends calling task until a particular character is received from the device.

Initialization. Triggered by activation of the CPU RESET line, initialization is the "housekeeping" that brings the system to the point where normal execution can begin, and then begins that execution. Table 4 lists the VRTX initialization system calls. The details of initialization inevitably vary from one application to the next, but the following description is representative.

When the CPU senses RESET, it invokes a user-supplied initialization routine (via a ROM-based vector in the 68000 family and by jumping to a predefined address in the iAPX86 family). The initialization routine

- initializes the memory subsystem (for example, MMU mapping registers), if necessary;
- writes the address of the configuration table and VRTX's entry point into the appropriate vector table entries (if not already encoded in ROM);
- issues the VRTX_INIT system call; (In response, VRTX initializes its internal variables and returns.)
- initializes the interrupt controller, timer, and serial I/O channel, as appropriate;
- issues VRTX system calls to create tasks, queues and memory partitions, and anything else that is needed to establish the initial execution environment; and
- issues the VRTX_GO system call to start normal execution of the highest priority ready task. (There is no return from this call.)

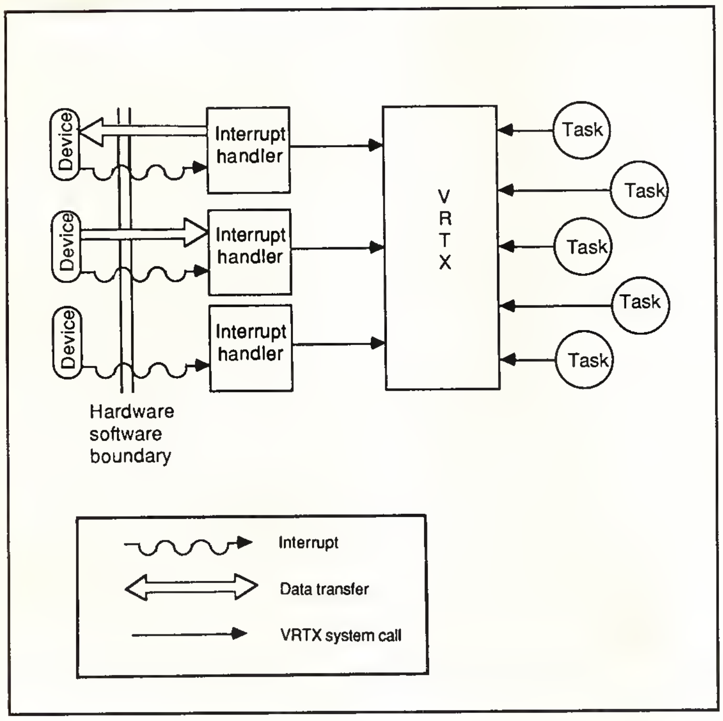
Figure 6 shows, schematically, how a VRTX-based software system looks after initialization is complete, in particular the way its various modules are linked together.

Real-time clock independence. Time in VRTX is made hardware independent by the notion of a clock tick; a tick is derived from an interrupt generated by any hardware timer. Table 5 lists VRTX's real-time clock system calls. VRTX maintains a clock counter that accumulates ticks; it increments the counter whenever an interrupt service routine (discussed later) issues the UI_TIMER system call. Application software can interrogate or reset the counter by other system calls. Tasks can schedule themselves to run at regular intervals simply by delaying themselves for the proper number of clock ticks.

Character device independence. Many applications incorporate a simple character-oriented I/O device (for example, a message display panel or operator keypad), and VRTX provides basic support for this type of device. VRTX provides a buffer between tasks and the interrupt handler that transmits to and receives from the device.

Two system calls enable tasks to get characters from the buffer and put them into it. Another call permits a task to suspend itself until a particular character arrives. Two calls used by interrupt handlers provide the device-independent interface for the VRTX character I/O channel. The UI_RXCHR call informs VRTX that an input character has arrived. The UI_TXRDY call operates whenever VRTX has a character available for output. In both cases the specifics of programming a particular character device are left to small interrupt routines outside of VRTX itself. The character I/O system calls are listed in Table 6.

Interrupt environment independence. In a VRTX-based system user-written interrupt handlers provide the interface



between tasks and devices (see Figure 7). When the CPU recognizes an interrupt, it immediately transfers control to the interrupt handler associated with the device. Thus, the interrupt handler is invoked directly by the hardware with no VRTX-imposed overhead; similarly, VRTX places no constraints on the operation of the interrupt handler. A typical handler, however, does three things; it

- resets the device,
- sends a message to notify a task of the interrupt (the message typically contains any data received from the device), and
- calls VRTX to reschedule (the interrupt may have readied a high-priority task) and to resume task execution.

When an interrupt occurs, two things must happen as quickly as possible. First, the task principally responsible for responding to stimuli from the device must be notified so that it can begin to formulate its response. Second, to minimize the chance of missing an interrupt, interrupts at the current and lower priority levels must be enabled. (The underlying microprocessor hardware design defines interrupt priority levels; they are not the same as VRTX task priorities.) Accordingly, interrupt handlers are usually implemented as short sections of assembly language code. Table 7 lists the calls VRTX provides for interrupt handlers.

Table 7.
Interrupt handler system calls.

System call	Definition	Action
UI_TIMER	Announce timer interrupt	Notifies VRTX that a tick has occurred.
UI_EXIT	Exit from interrupt	Returns from interrupt and causes rescheduling if necessary.

Table 8.
Extended task management calls.

Call	Action
TCREATE	VRTX calls this hook when it creates a new task; a register points to the new task's TCB. Typically, a TCREATE routine extends the TCB to hold additional application-specific information and initializes this information.
TDELETE	VRTX calls this hook when it is about to delete a task; a register points to the task's TCB. A typical TDELETE routine deallocates the TCB extension allocated by the TCREATE routine.
TSWAP	VRTX calls this hook whenever it performs a context switch (i.e., suspends one task and resumes another); registers point to the TCBs of both the old and the new tasks. The TSWAP routine typically updates information stored in the TCB extension or saves information into the old extension and loads information from the new extension.

Most other VRTX system calls can be issued by interrupt handlers as well. For example, an interrupt handler can pass data to a task with SC_POST or SC_QPOST.

Language and specialized device independence. VRTX provides core services, which virtually every embedded microprocessor application can use to advantage. Some applications have special requirements that are not addressed directly by VRTX. We have designed VRTX to accommodate such applications while preserving its software component orientation—VRTX never has to be modified. Note also that, because VRTX does not appropriate interrupts for itself, application-specific interrupt handlers can be written without restriction. A handler that services memory-refresh interrupts, for example, can be implemented completely outside of VRTX; it is invoked directly by the hardware and incurs no VRTX-imposed overhead—VRTX does not even know the interrupt handler exists.

Built into VRTX are three hooks to which application code can be attached if desired. The hooks are functionally equivalent to the "user exits" found in some operating systems and utilities; they are a mechanism for performing additional task-related processing without having to modify VRTX itself. Whenever VRTX reaches certain key points in its execution, it checks for the presence of an application routine; if it is present, VRTX calls the routine. When called, the routine receives access to a key VRTX data structure called a task control block, or TCB. The TCB holds a task's state, register contents, stack and instruction pointer values, and so on. The three hooks are described in Table 8.

These hooks are quite versatile, and VRTX imposes no constraints on their use. We list a few examples:

- *Performance measurement.* A TSWAP routine can maintain in a TCB extension a count of the time each task has used.
- *High-level language environments.* Some high-level language implementations have runtime environment values that must be saved and restored at task swap time to ensure reentrancy.
- *Floating-point units.* The TCREATE routine can allocate a TCB extension for saving the state of the floating-point unit and can initialize its state (for example, select 32- or 64-bit precision). The TSWAP code can save and restore the state of the floating-point unit, and the TDELETE code can deallocate the TCB extension.
- *Memory management units.* The TCREATE routine can initialize a TCB extension with address-mapping register values. The TSWAP routine can change the memory map using the new task's register values. The TDELETE routine can, as usual, deallocate the TCB extension.

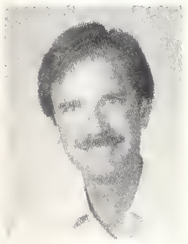
These components—versions of which are available for the Motorola 68000 family, the Intel 8086 and 286, the National Series 32000, and the Air Force standard architecture Mil-Std 1750A—provide the basis for handling the real-time deadlines and concurrent execution common in embedded systems. They are flexible enough to support a wide variety of unusual I/O devices and meet their deadlines. And finally, they support software developed on a wide variety of host computers with a number of high-level languages. The application software can easily be moved not only from different hardware based on a particular CPU family (for example, Motorola M68000) but also from one CPU type to another (that is, it is possible to recompile and move to an Intel 8086 environment from a National 32000 system).

References

1. MacLaren, Lee, "Evolving Toward Ada in Real-Time Systems," *Proc. ACM SIGPLAN Symp. on the Ada Language*, Dec. 1980.
2. Wirth, Niklaus, "Toward a Discipline of Real-Time Programming," *Comm. ACM*, Aug. 1977, Vol. 20, No. 8.

Bibliography

VRTX/68000 User's Guide, Hunter & Ready, Inc., Palo Alto, CA, 94306.



James F. Ready is senior vice president of Hunter & Ready, Incorporated. His research interests include real-time software design methodologies and the application of Ada to real-time embedded systems.

Ready received his BS in biological psychology from the University of Illinois, Urbana. His MA, also in biological psychology, is from the University of California, Berkeley. He is a member of IEEE and ACM.

Ready's address is Hunter & Ready, Inc., 445 Sherman Avenue, Palo Alto, CA 94306.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 156 Medium 157 Low 158

A network for a critical application keeps its performance high and its downtime low. How did its designers enable it to do both?

Performance and Availability in a Network File Server

William A. Jackson, Paul C. Rogers, Robert J. Hearn, and Jeffrey S. Mattiace
Electronic Data Systems Corporation

All computer users appreciate a system that offers superior performance and high availability. However, financial and technological constraints usually make it necessary to accept some degree of compromise. Some computer applications are less tolerant of such compromises than others. A patient-care information system, or PCIS, for a hospital is an example of a computer system in which compromises in performance and availability must be kept to a minimum. The medical professionals who depend on the PCIS legitimately expect a level of performance and availability consistent with the responsibilities they constantly face.

Here, we present an implementation of a central file server for a LAN-based PCIS. (File servers are surveyed by Svobodova,[1] and local-area networks by Stallings.[2]) We discuss the results of a project to move a central file server previously implemented on a 16-bit minicomputer to a current-generation 32-bit minicomputer (sometimes referred to as a "supermini"). The motivation for this project was the need to satisfy market pressures for increased system capabilities that exceeded the capacity of the earlier-generation 16-bit host. The objectives of the project were twofold:

• Increased performance. To meet the needs of larger hospitals, a central file server has to be capable of supporting a network of more than 400 workstations without decreasing response time at the user interface. It also has to be capable of supporting large databases—ones larger than eight gigabytes. A file server on a 16-bit host, however, can support only 60 to 70 workstations and a database of one gigabyte.

• Increased system availability. Reliable equipment and software are essential for high availability. Still, component failures are inevitable, especially in a large system. Redundancy of components in the system configuration can contribute to system availability by minimizing or eliminating downtime. And procedures for recovering from hardware or software faults—and the proper tools to implement those procedures—can play a vital role in increasing system availability. (For an examination of issues relating to component failures, see Date,[3] and for discussions of redundancy, see both Date and Kim.[4])

In many cases, the implementation choices which serve to increase system availability tend to have an adverse impact on system performance. The approach we took in these instances was to concentrate on the performance aspects of the algorithms used to effect the necessary availability improvements, and to apply compensating performance-improving techniques in other areas.

Many factors influenced the end result. Some performance improvement, for instance, can be credited to the labor expended to test and optimize routines for table searches and for memory (heap) management. Special microcoded instructions for queue management available in the host computer and an optimizing compiler each made favorable performance contributions. Hardware-imple-

The original Patient Care Information System

The Patient Care Information System, or PCIS, was first implemented in the mid-1970's. A Varian 16-bit minicomputer served as the central file server host for approximately 25 diskless workstations. The system was later moved to Univac and Data General 16-bit minis. In these implementations, the central file server process was written in assembly language and was limited to a 64-kilobyte address space.

A custom terminal control unit, or TCU, connected the minicomputer host to the local-area network. It contained its own microprocessor, which supervised LSI communications controllers to implement the LAN protocol. On-board DMA devices performed data transfers in either direction between the communications controllers and the TCU memory. Because the communications buffers in the TCU memory directly mapped into the minicomputer's address space, they were accessible by the central file server process.

The central file server supported a single TCU, which could in turn control from one to four communications lines. Each line could support approximately 25 workstations. While the line speed was relatively modest by today's standards—500 kilobits per second—separate transmit and receive DMA channels for each line enabled simultaneous transfers, and the burst rate over four lines could reach two megabits per second.

From the TCU each communications line passed to a line driver assembly, or LDA. The LDA contained individual circuits for each attached workstation. These circuits conditioned signals and electrically isolated the individual workstations from one another. They were designed to automatically disconnect a workstation from the network should the workstation fail in a constant transmit condition. A second TCU from a backup host could connect to the same LDA as the first TCU, and workstations could be manually switched, either individually or as a group, to either host. From the LDA, individual communications lines were routed radially to the workstations, which could be located up to 2000 feet away.

User workstations were based on the eight-bit Z80 microprocessor, which ran at 2.5 MHz. Thirty-two kilobytes of RAM provided space for application programs and a multitasking supervisor.

Application programs were interpreted pseudocode compiled from a high-level language. The supervisor and the application programs were loaded from the minicomputer host via the network. Network services were provided to applications via supervisor calls that were translated into service requests and forwarded to the central file server. Primary among these services was access to a common application database; other services—message routing between workstations, access to minicomputer peripherals such as high-speed tape drives and line printers, and time and date synchronization among workstations, for example—were also provided.

User workstations were equipped with integral badge readers to control access to the network. A user signed on to the network by inserting his badge into the reader; he signed off simply by taking his badge out of the reader. The badges permitted the system to limit a user's access to the authorized portions of the database and to restrict the use of the terminal to authorized applications.

Each workstation was also equipped with two serial ports. These allowed local printers, laboratory instruments, or other devices such as bar code readers or modems to be attached to the workstation. Because the application supervisor was multitasking, local printing or instrument monitoring could take place simultaneously with interactive user applications.

This original PCIS implementation is still in use at small- and medium-sized hospitals. A maximum configuration supports approximately 60 to 70 workstations and a database of up to one gigabyte. Limitations of the 16-bit host preclude the addition of most enhancements demanded by the marketplace. However, in 1982-83 we were able to upgrade the workstation by converting its CPU to a 6-MHz Z80B microprocessor, increasing its RAM to 64 kilobytes, and adding a math processor. These improvements, provided to meet demands for better performance, served only to whet the appetite of customers for an even better central file server host.

The 16-bit implementation influenced the design of the 32-bit one in another important way—the presence of an established customer base mandated that the new implementation be upward-compatible with earlier versions.

mented error-correcting memory capable of continuously "sniffing" for correctable errors contributed to system reliability, as did highly reliable, sealed (nonremovable) disk drives. Here, we will concentrate on those areas of the implementation that significantly influenced system performance and availability. Keep in mind, however, that the integration of the system components was in itself a significant factor influencing performance and availability, and that the many mundane aspects of the implementation combined to introduce system-wide complexities.

Performance features

The central file server host minicomputer. We chose the Data General MV series 32-bit minicomputer to be the host machine for the network's central file server. The MV series provides a family of computers suited to the performance needs of small, medium, and large hospitals. The very large real memory address space provided by every machine in the MV series (even the smallest, the MV4000) allowed us to provide all the features of the central file server to hospitals of any size. Indeed, the only differences between small and large installations are the number of workstations supported, the size of the PCIS database, and the amount of dynamically allocated memory used. The system is composed of the same program set for all installations.

The five members of the MV family are compared in Table 1. Larger members of the family achieve higher performance through memory caching, a higher memory and I/O bandwidth, a faster machine cycle time, increased pipelining, and additional peripheral capacity. The largest of the series, the MV20000, is available in both single-CPU and dual-CPU configurations. Both the MV10000 and MV20000 have multiple I/O channels.

The advanced instruction set of the MV minicomputers and the sophisticated features of Data General's AOS/VS operating system contributed to both the performance and the reliability of the central file server implementation. Not the least of the advantages provided by these 32-bit minicomputers over their 16-bit predecessors was, again, the very large address space available for each process. Without this very large address space, many of the features described below would have been impossible to implement.

A backup host for the network's central file server as well as a means to quickly switch the network from the primary host to the backup are provided. (These were also provided in the 16-bit implementation.) Dual-ported disk drives with intelligent controllers are physically attached to each host

Table 1.
Data General MV series—family comparison.

	MV4000	MV8000-II	MV10000	MV20000-1	MV20000-2
Relative performance	0.5	1.0	2.0	4.0	7.0
I/O bandwidth (M bytes/s)	5	18	28	35	35
CPU/memory bandwidth (M bytes/s)	10	18.2	28.6	47	47
Machine cycle time (ns)	200	220	140	85	85
Max memory (M bytes)	8	8	32	64	64
Data cache (K bytes)	None	16	16	16	16/CPU
Instruction cache (K bytes)	None	None	4	4	4/CPU
Address translation cache (K bytes)	4	4	4	4	4/CPU
Max disk storage (G bytes)	9.4	14.2	27	27	27

machine. Other peripherals such as tape drives and line printers are either duplicated or made switchable. More on the PCIS network appears in "Network overview."

We designed a high-performance, custom network interface that is installed in the backplane of the MV series computer. This interface—which we call the terminal control

Network overview

PCIS applications execute on diskless microprocessor-based workstations. The workstations are connected via a local-area network to a minicomputer that hosts the network's central file server. One or more custom controllers called terminal control units, or TCUs, reside in the minicomputer's backplane and provide the physical network interface between the workstations and the minicomputer host. A burst multiplexer channel interface, or BMCI, provides high-speed (six-megabyte-per-second) data transfers between the host and each TCU.

Each TCU contains its own microprocessor and memory for communications buffers, and each handles the link-level network protocol for four synchronous communications lines. Each line connects through a line driver assembly, or LDA, to a maximum of 42 workstations. Workstations can be located up to 2000 feet from the LDA. The LDA provides electrical isolation, signal conditioning, and switching capability to the individual workstations.

Each installation has a primary and a backup minicomputer host. The LDA allows the workstations to be manually switched, either individually or as a group, from the TCU on the primary host to a corresponding TCU on the backup host. Dual-ported disk drives are attached to a disk controller on each minicomputer, although they are logically on line to only one machine at a time. Other minicomputer peripherals such as tape drives and line printers are either duplicated or switchable between hosts.

The central file server acts as an extension of the workstation's operating system. Through supervisor calls, application programs executing on a workstation gain access to the PCIS database and obtain other services provided by the central file server. These include access to minicomputer peripherals, program loading, and message routing between workstations.

Figure A is a generalized diagram of the PCIS network.

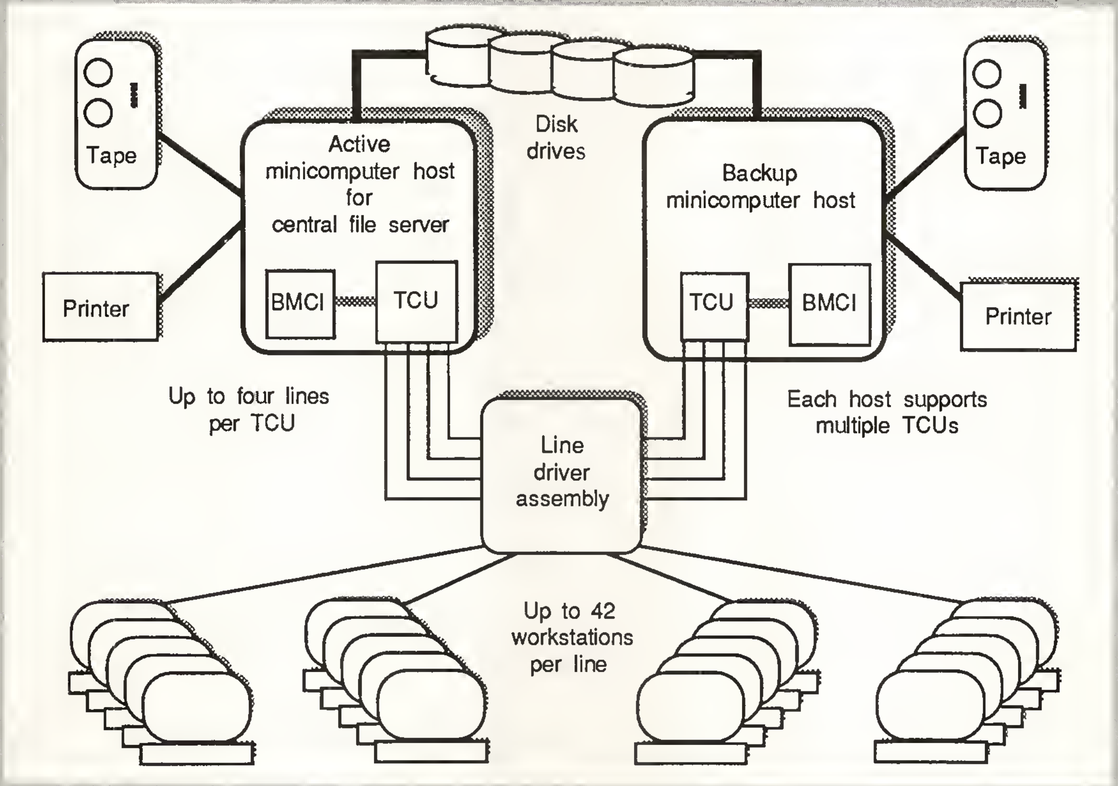


Figure A. The PCIS network.

unit, or TCU—features its own microprocessor and local memory. Its memory consists of two physical partitions, one for communications buffers, another for a control program. The control program is loaded into the TCU by the central file server at start-up. LSI communications controllers and DMA devices enable the TCU to handle the link-level network protocol with ease. Packet transfers between the TCU and the host minicomputer take place over a high-speed (six-megabyte-per-second) burst multiplexer channel interface, or BMCI. A detailed description of the TCU appears in "Terminal control unit," below.

Disk caching. Perhaps the most significant single contribution to software performance is the implementation of disk caching. In our system, the system administrator may select any of the files of the PCIS database for disk caching. For each file selected, a specified number of record buffers is allocated from the free memory (heap) area of the central file server. These buffers are used to hold the most recently read records from the file. The buffers are chained together in a doubly linked list, allowing them to be efficiently searched and manipulated with a special set of microcoded queue management machine instructions available on the MV series computers. When a read request from the network specifies a file that is cached, the central file server first searches the file's cache buffers for the record. If the record is in the cache, the read request can be satisfied without a disk access. When a record must be read from the

Terminal control unit

Electronic Data Systems worked with Data General's Special Systems Division to design the terminal control unit (Figure B). The TCU, along with a specially adapted Data General burst multiplexer channel interface (BMCI), provides high-speed DMA transfers between itself and the host computer's memory and can support simultaneous full-duplex communications at rates of up to one megabit per second on each of four synchronous serial I/O ports. To achieve this performance, each TCU serial I/O port is equipped with two dedicated DMA channels, one for transmit and one for receive. A 64-kilobyte block of multiported, 80-nanosecond memory on the TCU is reserved for communications buffers. An 8-MHz 8088 microprocessor supervises the operation of the serial ports and works with the host computer to set up BMCI transfers between the TCU's communications buffers and the host's main memory. The TCU and the host exchange command and status information through six control registers in the I/O device address space of each. Each can interrupt the other to signal the initiation or completion of a significant operation.

A boot ROM on the TCU contains code that performs self-diagnostics at power on and can perform more extensive diagnostic tests under the control of a host-resident routine. Other ROM-resident routines are used to download a network control program from the minicomputer host. The control program is loaded into a 32-kilobyte block of program memory on the TCU. This program memory is also used to hold the I/O queues and control tables used in implementing the network communications protocol and in transferring data between the TCU and host. Because the bus for program memory is isolated from the bus for the communications buffer, the normal instruction fetches and data accesses of the TCU's microprocessor do not interfere with DMA transfers to and from the communications buffer. However, the microprocessor can still access data in the communications buffer if it needs to. But it seldom does, since its primary objective is to transfer information packets between the serial communications ports and the main storage of the host and since DMA channels are provided to perform these transfers.

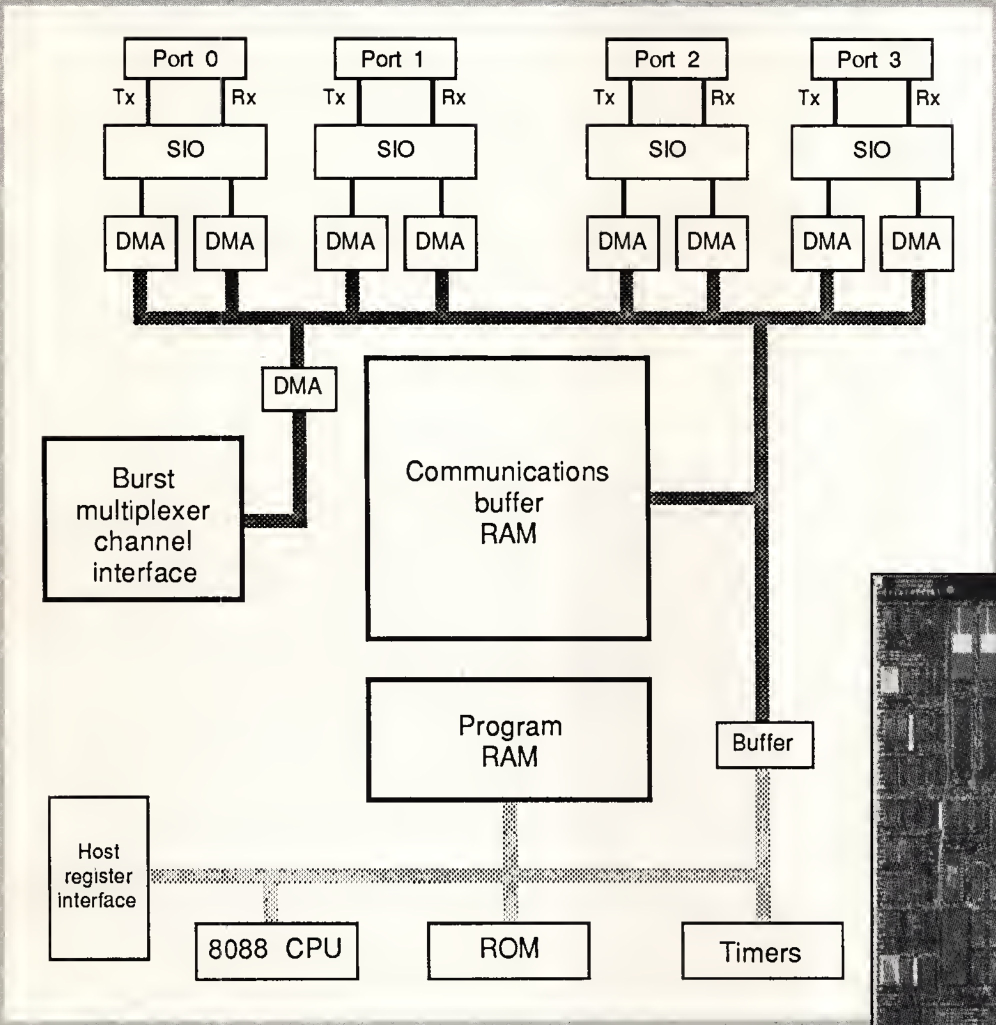
The control program on the TCU cooperates with routines of the central file server process to perform high-speed block transfers of message packets between the TCU's communications buffer and the minicomputer's main storage. The special version of the Data General BMCI was produced to support these block transfers. (The standard version of the BMCI is typically used to provide a high-speed data path into memory for block transfer devices such as disk drives.) The BMCI's path width is 16 bits, and transfers over it normally take place in increments of sixteen 16-bit words so that multiple devices can multiplex their block transfers in short bursts. Dual 16-word FIFO buffers compensate for the multiplexing delays that occur when several devices are contending for the data path. Under test conditions, sustained data transfer rates exceeding 4.5 megabytes per second have been obtained between the TCU's communications buffer and the minicomputer's main storage while bidirectional transfers of 2048-byte blocks over the BMCI are being performed. This figure includes time for setup, interrupt service, and the status checking of individual blocks. A single BMCI can interface up to four TCUs to the minicomputer, and multiple BMCIs can be installed to handle very large networks.

disk file, that record replaces the least recently accessed record in the cache.

Disk caching can be likened to the memory caching that is used to shorten effective memory access times on CPUs. However, the impact of disk caching can be more dramatic. For example, a memory cache hit may reduce the time for the CPU to fetch a word from main memory from 350 nanoseconds to 40 nanoseconds—a one-order-of-magnitude savings. But a disk cache hit can reduce the time to fetch a record from 40 milliseconds to 40 microseconds—a three-order-of-magnitude savings.

Cache hit ratios of 80 to 95 percent are not unusual for some files in the PCIS database. In one exceptional case, a performance report indicated that a file with a 20-record cache buffer had been accessed by 125,173 read requests within a 30-minute period. Of these, 124,439 requests had been satisfied from the cache—a 99.4-percent hit ratio. Notice that this represents an average service rate of almost 70 records per second, or about one record every 14 milliseconds. The 738 actual disk accesses to this file (four of the accesses had been write requests) represented three percent of the total disk accesses by the central file server during the same period. It is unlikely that such a service rate could have been maintained without the disk caching facility.

Disk caching is costly in terms of memory—a 20-record cache for a file with 2048-byte records requires 40 kilobytes of main storage. It also requires additional CPU overhead for cache searches, for both read and write requests. Thus,



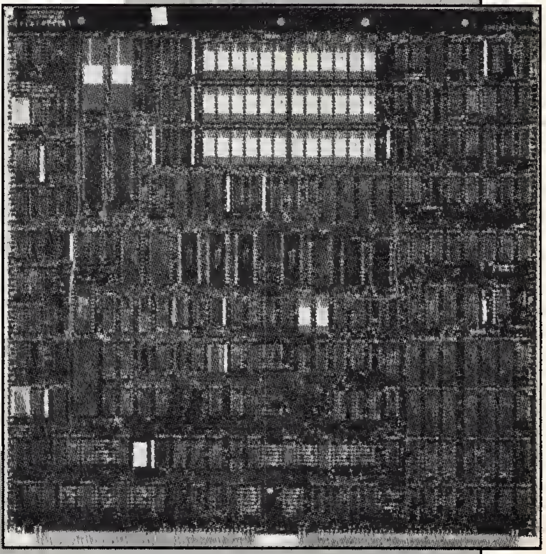


Figure B. Block diagram of the terminal control unit. The inset shows the actual TCU board that resides in the backplane of the Data General MV series computer.

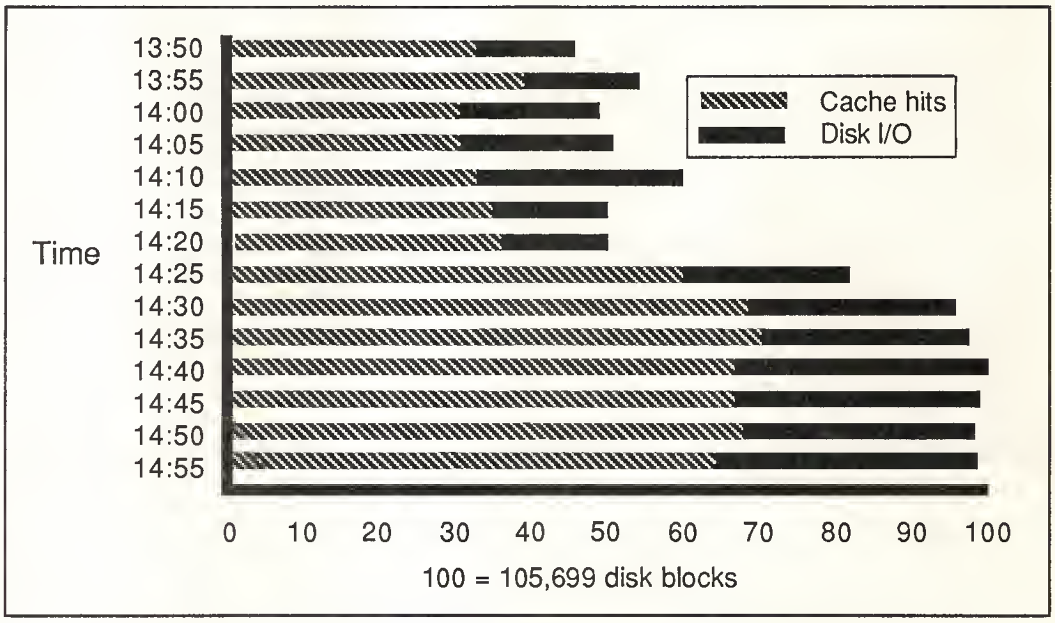


Figure 1. Effect of disk caching on database activity.

disk caching should not be used indiscriminately. Highly active files with a moderate to high read-to-write ratio are good candidates for caching. The best candidates also exhibit a small locality of access. (This locality can drift slowly over the entire file, however.) Within the PCIS database, two file types exhibit exceptionally good characteristics for caching. These are program files containing executable workstation load modules and files containing predefined menus or screen images used by the PCIS application programs. Because the number of record buffers allocated can be individually specified for each file, the system administrator can distribute the available memory among different files to maximize the benefit of the disk caching facility.

We monitored the effect of disk caching on database activity on an actual PCIS network (Figure 1). The host was

The central file server process

The central file server executes as a high-priority privileged process under Data General's AOS/VS operating system. As a privileged process, it can perform many functions normally reserved as operating system functions, yet it can take advantage of all the normal AOS/VS services. In the first category are functions such as direct access to and control of the TCU. The central file server accesses the TCU as if it were an I/O device. It services TCU interrupts directly. Also included in this category is the locking of virtual memory pages into physical memory. With the exception of some code executed during start-up, the process locks all of its pages into real memory to avoid paging delays during the servicing of network requests.

Because the central file server runs as a process under AOS/VS, other processes can share the machine on a time-available basis and can communicate with the central file server through the interprocess communications facility of AOS/VS. This capability is important to ancillary functions related to the support of PCIS, e.g., database backup/restore functions, batch programs that process input or output for foreign systems, and other administrative functions. AOS/VS also provides a well-developed file system, numerous utilities, and some performance monitoring tools, all of which can be used to support PCIS.

Two interrupt service routines are a part of the central file server process. One handles interrupts from the TCU. TCU interrupts indicate that the TCU has completed some service requested by the central file server and is presenting status information, or that the TCU has a packet from the network ready to be transferred to the central file server. The central file server's other interrupt service routine handles interrupts from the burst multiplexer channel interface. The central file server and the TCU-resident control program work together to set up bidirectional transfers of network packets over the BMCI. Interrupts from the BMCI signify the completion of such transfers.

an MV10000; 197 workstations were active on the network during at least some portion of the period monitored. We checked the total effective database activity (cache hits plus physical I/O) every five minutes over a period of 70 minutes. The mean effective throughput was 154.9 blocks per second during the least active interval and 352.2 blocks per second during the most active interval. For each interval, at least 64 percent, and at most 75 percent, of the total activity was satisfied by cache hits.

A general discussion of the effect of disk caching on performance is provided by Friedman.[5]

Multitasking. Data General's AOS/VS operating system permits a process to have multiple I/O operations outstanding only if the process is multitasking; a typical process implemented as a single task is suspended by AOS/VS when it initiates an I/O operation and remains suspended until the I/O operation is complete. AOS/VS allows a multitasking user process to have up to 32 active tasks, and each of the tasks can have an I/O operation in progress at a given time. This permits the process to overlap I/O operations, and to continue CPU operations even while multiple I/O requests are pending.

The central file server executes as a high-priority, privileged user process under AOS/VS. It takes advantage of AOS/VS's multitasking facility by activating 30 separate tasks. A master scheduler task performs all the work needed to service a network request that does not require a system call that can result in the task being suspended. When it is likely that continued processing of a request will cause the processing task to be suspended, the request is passed to one of 29 specific service tasks, and the master scheduler continues by retrieving another request from its own work queue. Of the 29 specific service tasks, 20 are designated for database service. The remaining are for peripheral access, network service, critical event logging, statistics logging, interprocess communications, and maintenance of the network's master time clock. Requests for service are passed between the tasks via work queues assigned to each. More information on the central file server process appears in the box at left; the task and queue structure is shown in Figure 2.

The multitasking facility is also used to advantage in the PCIS system's backup and restore utility. By using separate tasks for reading and writing—and large I/O buffers to support multisector reads and writes—PCIS backup and restore operations take much less time than single-task backup and restore operations would. These faster operations directly contribute to availability by reducing downtime when a restore is needed. They also reduce the amount of time the system administrator must devote to routine backup operations.

Locked memory pages. AOS/VS is a virtual memory operating system. To ensure that requests from the network are serviced as quickly as possible, the central file server process locks virtual memory pages into real memory. This avoids delays caused by paging and eliminates some overhead that would otherwise be encountered in transferring packets between the process's memory and the communications buffers of the network interface. With the exception of some code executed during start-up, all allocated pages are locked. The amount of real memory required depends on the number of workstations supported, the number of files in the PCIS database, and the extent to which the disk caching and atomic transaction support facilities are used. In most cases, a central file server utilizes from two to four megabytes of real memory.

Integrated performance monitoring. Performance monitoring and reporting are needed to verify that performance goals are met and to ensure that the system continues to meet those goals while operating in a dynamic environment. These tools also are useful in understanding the behavior of the system, and thus they help to ensure that changes made to the system are based on knowledge rather than guesses. The routines of the central file server have built-in facilities for collecting statistics related to the performance of the process. The statistics that are collected fall into four categories: internal work queues, database access, log activity, and network activity.

Each of the task work queues within the central file server process is associated with two counters. The first reflects the number of elements currently pending on the queue; the second contains the maximum number of elements that have been in the queue since the process was activated. Information provided by the counters can help identify potential performance bottlenecks, either with I/O or other internal processes.

For each file in the PCIS database, counters are provided for accumulating the number of disk reads and the number of disk writes. If the file is dual-imaged (dual imaging will be discussed in the section on availability features, below), separate counters are assigned to each image. For cached files, a counter is allocated to accumulate the number of cache hits.

Additional counters monitor the writes to the critical events log (also described below) as well as the buffer management activities performed by the logging process.

Network activity is measured by counting the the number of different types of request serviced for each workstation on the network and the number of data units transferred for each request.

One can monitor these statistics in real time by requesting the central file server to report the contents of selected counters. Interactive programs that display, in either graphical or tabular form, the dynamic relationships of various aspects of system behavior can be run on any workstation. In Figure 3, for example, database activity is being dynamically displayed. The bar graph depicts the relative proportion of the current activity by disk drive. Columns to the left of the bar graph identify the disk drives and give the number of accesses during the current sample interval as an absolute value and as a percentage of total activity. The rightmost column shows the average activity of each disk drive since the central file server was activated. These percentages are also reflected by the position of the arrows

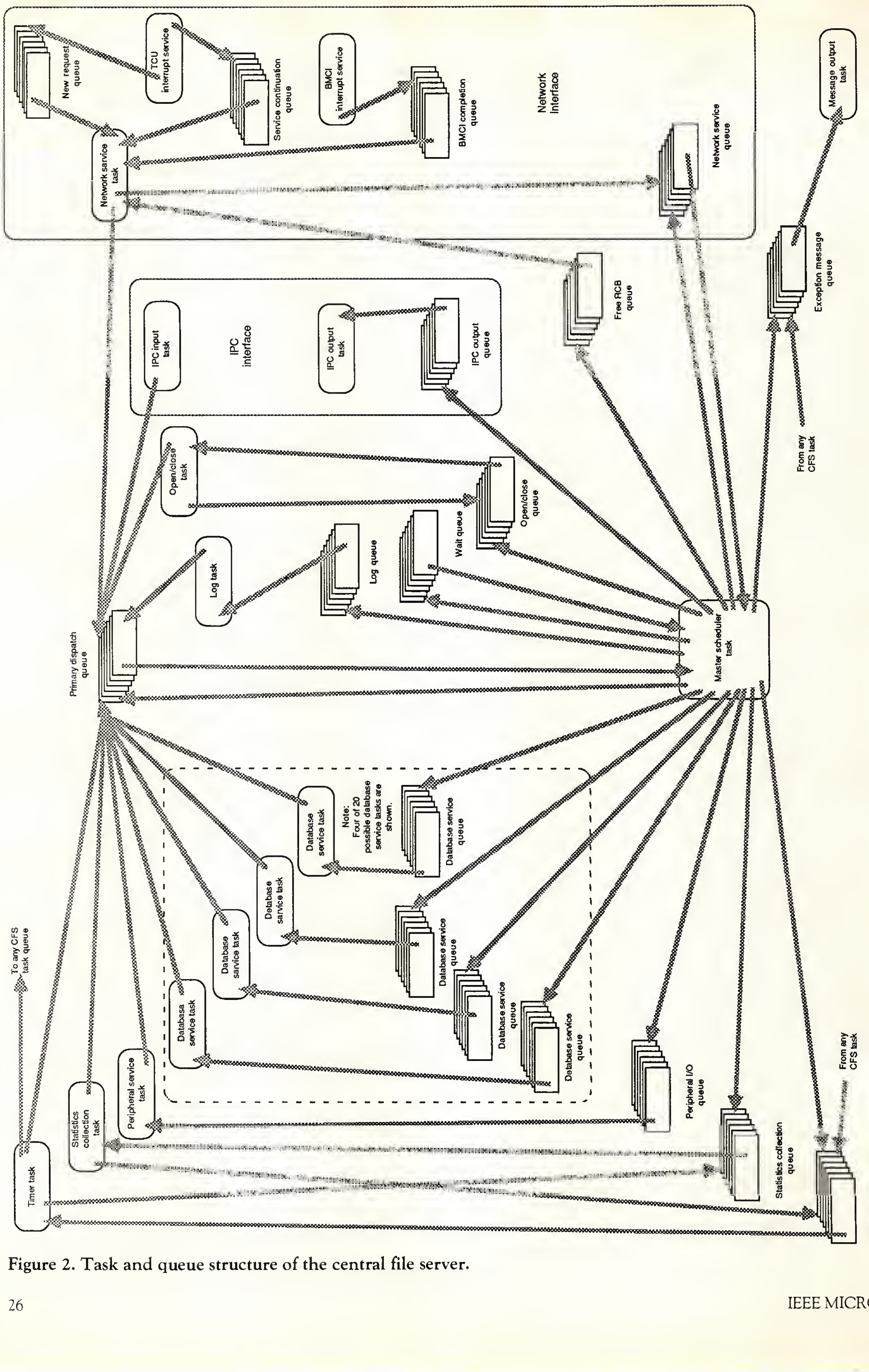
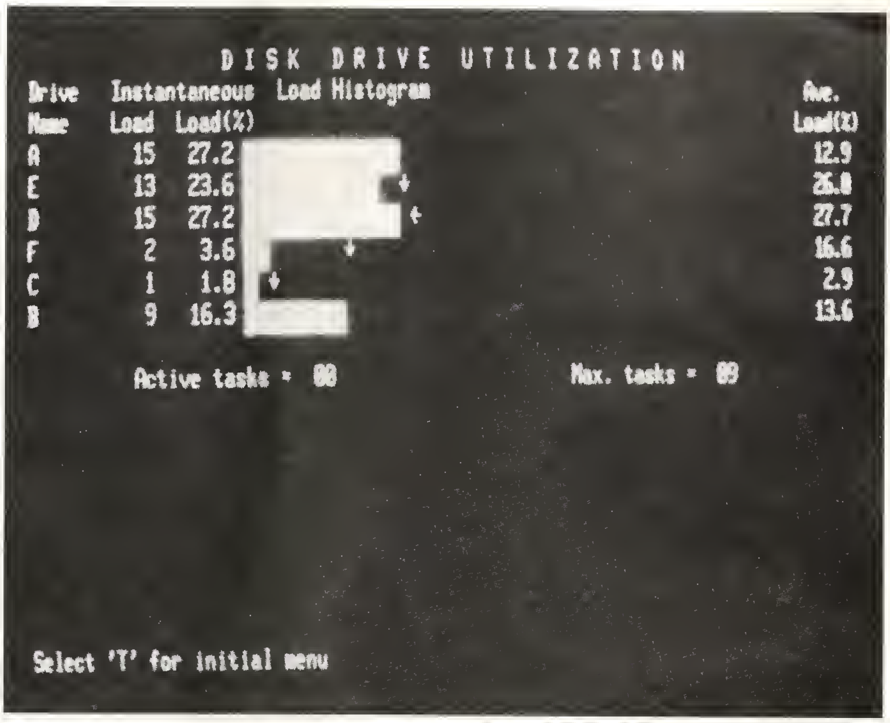


Figure 2. Task and queue structure of the central file server.



```
                 D I S K   D R I V E   U T I L I Z A T I O N
Drive   Instantaneous  Load Histogram                              Ave.
Name    Load  Load(%)                                              Load(%)
A         15   27.2                                                  12.9
E         13   23.6                                                  26.8
D         15   27.2                                                  27.7
F          2    3.6                                                  16.6
C          1    1.8                                                   2.9
B          9   16.3                                                  13.6

           Active tasks = 08                     Max. tasks = 09




Select 'T' for initial menu
```

Figure 3. Monitoring of system performance at the workstation—here, the display shows disk drive utilization.

that can be seen on the graph. The line below the graph displays the number of currently active database service tasks and the maximum number of these tasks that have been concurrently active since start-up. Since the system administrator can transfer database files from one disk drive to another to achieve a better balance of disk activity, this display can help him check the results of his action.

One can also activate a task within the central file server that periodically records the contents of all accumulators on disk. One then runs a program that produces chronological profiles of system activity from the recorded statistics.

This program employs established rules to make recommendations for eliminating potential performance bottlenecks or misused resources detected through the analysis of the data. For instance, the program may recommend that disk caching for a particular file be eliminated even though the percentage of cache hits is high, because the frequency of reads is too low to justify the extra memory required by caching. Or it may recommend that a very active file be moved to another disk drive to help balance the activity across all drives. It may also recommend a configuration of the workstations that will improve throughput on the network.

Tables 2 and 3 illustrate how the program's recommendations can be used. Table 2 shows the 10 files of the PCIS database that were the most active over a 1.28-hour period. The two most active entries were the dual images of the file OELTXELS. Accesses to this file represented 34.1 percent of the total disk activity related to the PCIS database over the monitored period. Note that dual imaging spread this activity over two disk drives. Table 3 shows the 10 most active files over a later period of 1.16 hours, after a cache buffer of 30 records had been allocated for OELTXELS in response to the program's recommendation. The first image of the file has moved from first to second place and represents 6.2 percent of the total disk activity. The second image of OELTXELS has moved to 16th place and hence is not shown; it represents 1.9 percent of the total disk activity. During the 1.16-hour monitoring period, there were 11,301 disk accesses to the file, of which 7751 were reads. 4166 potential accesses were avoided as a result of hits in the cache buffer.

Note also the recommendation in Table 2 to move the file S#PRTQ4 to another drive. This is an example of a recommendation that helps the system administrator balance disk activity among available drives.

In addition to the tools described above, the AOS/VS operating system is equipped with generic tools for monitoring the utilization of both system-wide and process-specific resources. These tools help the system administrator maintain peak performance on the PCIS network.

Table 2.
The 10 most active disk files of the PCIS database over a 1.28-hour period.

Rank	File name	Drive ident	Number of accesses	% drive activity	% total activity	Cache size	File image	Comments and recommendations
1	OELTXELS	Y1D	36995	78.7	21.7	None	1	Very high, try caching
2	OELTXELS	Y2D	21235	78.2	12.4	None	2	High, try caching
3	S#PRTQ4	Y1A	8256	39.9	4.8	None	-	Med, move to drive Y0B
4	ADD02ALM	Y3C	6388	55.4	3.7	25	1	Med
5	S#UTIL	SYS	5673	60.0	3.3	None	-	Med
6	ADD018LS	Y1A	5039	24.4	2.9	None	1	Med
7	OEDTXALM	Y3C	4121	35.8	2.4	40	1	Low
8	PRD010RS	Y4B	3792	38.6	2.2	20	1	Low
9	SCREENRS	Y3A	3704	44.6	2.1	100	1	Low
10	OELTXRLS	Y1D	3562	7.5	2.0	None	1	Low

Table 3.
The 10 most active disk files of the PCIS database over a later, 1.16-hour period.

Rank	File name	Drive ident	Number of accesses	% drive activity	% total activity	Cache size	File image	Comments and recommendations
1	S#PRTQ4	Y1A	11617	55.8	8.4	None	-	Med
2	OELTXELS	Y1D	8598	54.0	6.2	30	1	Med
3	TSD01ALM	Y0A	6540	94.6	4.7	None	1	Med
4	S#SYST	SYS	5181	83.1	3.7	None	-	Med
5	ADD02ALM	Y3C	4972	46.4	3.6	35	1	Med
6	TSD01BLM	Y2D	4952	39.4	3.6	None	1	Med
7	SCREENRS	Y3A	4427	41.5	3.2	125	1	Med
8	OEDTXALM	Y2C	4189	39.1	3.0	40	1	Low
9	TSD01ALM	Y2C	4164	63.1	3.0	None	1	Low
10	ADD02ALM	Y1C	4058	43.7	2.9	35	2	Low

Availability features

Availability is improved by reducing downtime. Reliability is a key component of availability; it determines how frequently a system can be expected to fail. Another component of availability, hereafter referred to as recoverability,* determines how rapidly the system can be restored to operational condition after a failure. Improvements to either of these two components reduce expected downtime and thus improve availability.

The PCIS system is composed of hardware and software components, all influencing system reliability and recoverability. For us, influencing hardware reliability was pretty much limited to selecting reliable components and utilizing them correctly. Controlling hardware recoverability included providing redundancy in the configuration, providing correct procedures for utilizing the redundancy when required, and ensuring that an adequate service plan was available from the equipment manufacturers.

Software reliability depends largely on correct code.

* *Maintainability* is a more familiar term, but in the current context *recoverability* provides more appropriate connotations.

Some steps taken to ensure correct code were to employ a high-level language for programming the central file server* and to utilize the excellent debugging and source-code maintenance tools available with the host operating system. Software reliability also depends on ensuring that a system's routines can deal with transient system errors. We attempted to ensure that our routines could.

Software contributions to the improvement of availability were made chiefly in the area of recoverability. By automating some of the recovery processes, by designing the system's software so it provides reliable status information, and by providing tools the system administrator can use to aid manual recovery processes, we reduced the time the system needs to recover from a failure. We improved the quality of the system administrator's interface, giving him greater confidence and less anxiety in dealing with system failures. This improvement indirectly reduced system downtime by attacking the source of many human errors that otherwise protract the recovery process.

Dual-imaged files. Because the PCIS database is a critical resource that resides on disk, the failure of a single disk drive can bring the entire system down. In this case, redundant minicomputer hosts or spare workstations provide no help. Even with a spare disk drive, recovery would require that the system be unavailable during a lengthy recovery process. Dual imaging of the PCIS database files is a technique used to avoid this undesirable situation. (Dual imaging is discussed by Date[3] and Bates and TeGrotenhuis.[6]) The system administrator can select any file of the database for dual imaging. For optimum protection, all files of the database should be selected. Two images of each selected file are created. The two images must reside on different disk drives; preferably, each is on a drive attached to a separate controller. The central file server then ensures concurrency of updates to the two images. Proper distribution of dual-imaged files across available disk drives and controllers can mean that the failure of any single disk drive or controller will not noticeably affect users of the network.

The central file server handles network write requests to a dual-imaged file by simultaneously dispatching the request to the work queues of two separate tasks. When both writes have completed, a completion code is returned to the requesting workstation. A bad completion code is returned only if both writes fail. If the write to one image fails while the write to the other succeeds, the image on which the failure occurred is marked as being off line to the network, since that image is no longer current. No indication that a failure has occurred is given to workstations on the network; application programs continue to access the file via the remaining "good" image.

The failure of a disk drive will cause a write directed to any of the file images on the drive to fail. Consequently, it is likely that console error messages for several files will appear in quick succession. If the drive failure is such that the recording medium is not damaged (e.g., a power supply failure), repairs can be effected and full dual-imaged capability restored without workstation users becoming aware of the problem. Facilities within the central file server enable the system administrator to close all images on the failed drive. This permits the drive to be logically disconnected from the system and powered down for service. After repairs are completed, the drive is logically reconnected to AOS/VS. The central file server can then be instructed to reopen all images on the drive with writes enabled but reads inhibited. Then, for each file on the drive, the central file server can restore the "bad" image by copying blocks from the "good" image. As the copying operation completes successfully for each file, the read inhibit for the previously bad image is automatically removed. Full network access to the file is maintained even during this copying operation.

If a disk controller fails, or if a drive failure results in damage to the medium, the system probably will have to be brought down at some point in the recovery process. Even if this is the case, however, the ability to continue operation with only one image of the affected files allows the system administrator to muster resources for recovery and to schedule downtime to minimize its impact on network users.

Naturally, the additional write required for each update of a dual-imaged file means additional work for the central file server. The scheduling of reads to a dual-imaged file also creates some overhead. In spite of these, a system with dual-imaged files usually exhibits better performance at the workstation than a system without dual-imaged files. (This is confirmed by Date,[3] Bates and TeGrotenhuis,[6] and Wilkinson and Lai.[7]) This occurs because all reads (and writes) to a file with a single image are handled by a single task and are thus forced to queue up during periods of high activity. For dual-imaged files, the central file server can direct read requests to the image whose task has the smallest work queue. Two reads can be outstanding to the same file—one to each image. The result of these capabilities is faster servicing of reads to highly active, dual-imaged files. Because most of the files in the PCIS database exhibit a high read-to-write ratio, this faster servicing of reads provides a throughput advantage even though dual imaging requires twice as many writes to be performed.

Critical event logging. System shutdowns are either scheduled or unscheduled. A scheduled shutdown is initiated by the system administrator and is performed in an orderly fashion that assists recovery. First, the central file server refuses to allow new applications to be started on the workstations but permits currently active applications to complete. Next, it refuses new requests for service but allows pending transactions to complete. Finally, it halts network activity and closes the database files. The scheduled shutdown permits the database and message queues to be in a known state when processing is resumed.

In contrast, an unscheduled shutdown—such as one due to the failure of the system's primary power source—interrupts pending transactions suddenly; the system has no time

* Previous implementations of the central file server were coded in assembly language; the current version is coded in C.

to "tidy up" the database. In an active system, an unscheduled shutdown could probably cause some portion of that system to be left in a state that would produce application errors when it was restarted, if no means of dealing with unscheduled shutdowns were provided.

A restart procedure incorporated in the central file server enables the system to automatically recover from unscheduled shutdowns, however. This procedure requires that a disk-resident chronological log of certain critical events be maintained by the central file server during normal operation.

Any change to the PCIS database is considered a critical event. A write request from the network is a critical event, for example. Before the write request is applied to the database, it is first recorded in the log. The data to be written to the database are recorded, as are the address of the requesting workstation and other identifying information. Other critical events include changes to the network messages queues, the initiation and completion of any on-line backup or restore operation, the allocation or release of records in a free-record queue, and the initiation or completion of atomic transactions. Furthermore, a checkpoint of the central file server's resource control table is written to the log at predefined intervals (typically, every five minutes). A field within each checkpoint record contains a pointer to the position within the log of the previous checkpoint record. The latest checkpoint record is also written to a single-record checkpoint file, along with a pointer to the position within the log of its duplicate.

The central file server begins the restart procedure by opening all the files of the PCIS database. It reads the checkpoint file and uses it as the initial version of the resource table. The checkpoint file also allows the start-up routine to find the position in the log file of the last checkpoint record. The checkpoint record indicates whether the prior shutdown was scheduled (controlled). If so, no recovery procedures are necessary, since the state of the database and resource table is such that the network interface can be enabled and normal processing begun. Critical event logging can resume at the next record position in the log.

When the system is being restarted from an unscheduled shutdown, however, the pointer within the checkpoint record is used to step back through the log to the next previous checkpoint record. The start-up routine then moves forward toward the end of the log, reading each record and using the event represented to update the database or resource table as required. After the start-up routine has finished processing the last event on the log, logging resumes with a start-up checkpoint record, the network interface is enabled, and the central file server is ready to service network requests.

The work queue of the log task is very likely to become a system performance bottleneck. To reduce this likelihood, we expended a great deal of effort in optimizing the log task routines and in reducing the number of physical writes to the log file. (Some of our efforts toward these ends are described in "Queue element purging by the log task," at right.) We selected the characteristics of the log file, as they were to be described to AOS/VS, to optimize performance. While it is not absolutely necessary, we recommended that users assign an entire disk drive to the log file. If they can do this, the greatest time consumer, disk seeks, can be reduced to a single-track seek each time a cylinder fills.

Atomic transaction support. An atomic transaction consists of a set of updates to the database that must always be applied as a set. If some, but not all, members of the set are applied, the database is corrupted. Atomic transaction support guarantees that either all of the updates are applied or none of them are applied. The current implementation of the central file server provides new network request types for atomic transaction support. (Atomic transactions are discussed by Svobodova,[1] Ciompi et al.,[8] and Reed.[9])

To begin an atomic transaction, an application sends a start request to the central file server. The central file server logs the start request and reserves an atomic transaction queue for the requesting application. Then, as the application proceeds with the transaction, each request to update the database is written to the critical event log and placed in the atomic transaction queue, which is memory-resident. After the application has requested the final update of the atomic transaction set, it sends an end request to the central file server. The central file server then begins applying all of the updates held in the application's atomic transaction queue to the database. After it has applied all the updates, it releases the atomic transaction queue.

If an unscheduled shutdown occurs while an atomic update is in progress, the start-up recovery procedure will read the record of the start request from the log and begin reconstructing the memory-resident atomic transaction queue for the requesting application. As the start-up procedure encounters each update request of the atomic transaction on the log, it places it in the appropriate queue. If it encounters a record of an end request on the log, it applies all the updates in the corresponding queue to the database. If the start-up procedure reads the last logged event without finding an end request, then it purges the updates instead of writing them to the database. In this event, it issues a message identifying the workstation, the application, and other pertinent information so the purged transaction can be re-entered at the workstation. Note that if the unscheduled shutdown occurs *after* the end request is logged, *all* updates of the set are applied, and that if the unscheduled shutdown occurs *before* the end request is logged, *none* of the updates are applied.

To ensure that the large memory resources used by the memory-resident atomic transaction queues are not wasted, we provided for a time limit on atomic transactions. The central file server will automatically abort an atomic transaction if the time between the start and end requests exceeds this limit. Typically 30 to 45 seconds, this time limit can be adjusted by the system administrator at system start-up. The checkpoint interval (also controlled by the system administrator) should be at least twice the atomic transaction time limit to ensure that recovery from unscheduled shut-

Queue element purging by the log task

A critical event delays the processing of the network request associated with it until a record of the event is written to the disk-resident critical events log. We wanted to minimize this delay. Because data must be written to disk in multiples of 512 bytes, a special procedure was needed. The procedure we provided reduces the number of elements in the log task's work queue, and hence the number of scheduled disk writes to the log file. This procedure is illustrated in Figure C.

In Figure C1, the log task finds an element representing a record of a loggable event in the log queue. The record has been formatted into the one-kilobyte log buffer by the master scheduler task. In Figure C2, the element is removed from the queue, and a write system call is performed to write disk block 2330 (a 512-byte sector) to the disk-resident log file. This causes the log task to suspend and wait for the I/O operation to complete. While the log task is suspended, the master scheduler task gains control and formats additional log records into the buffer represented by log queue elements 2, 3, 4, and 5. In Figure C3, the log task becomes active, extracts the next element, 2, from the log queue, and prepares to perform a disk write. Before the write system call is issued, however, the remainder of the queue is scanned to see if there are additional records already formatted into a contiguous section of the buffer. Elements representing records 3, 4, and 5 are found and their queue elements are marked. The write system call is issued, this time for two disk blocks (2330 and 2331) because record 5 occupies a portion of both blocks. While the log task is suspended for this write, the master scheduler formats records represented by elements 6 and 7 into the buffer. On regaining control after the write, the log task purges the previously marked queue elements and begins processing element 6. In Figure C4, element 7 is marked for purging, and block 2331 is written to disk. Element 8 is placed on the log queue during the I/O operation.

Queue element purging by the log task eliminates some 1 to 10 percent (depending on the frequency of loggable events) of the scheduled disk writes to the log file. Defining larger buffers for the log task (an option available to the system administrator) allows this procedure to eliminate an even higher percentage of scheduled disk writes.

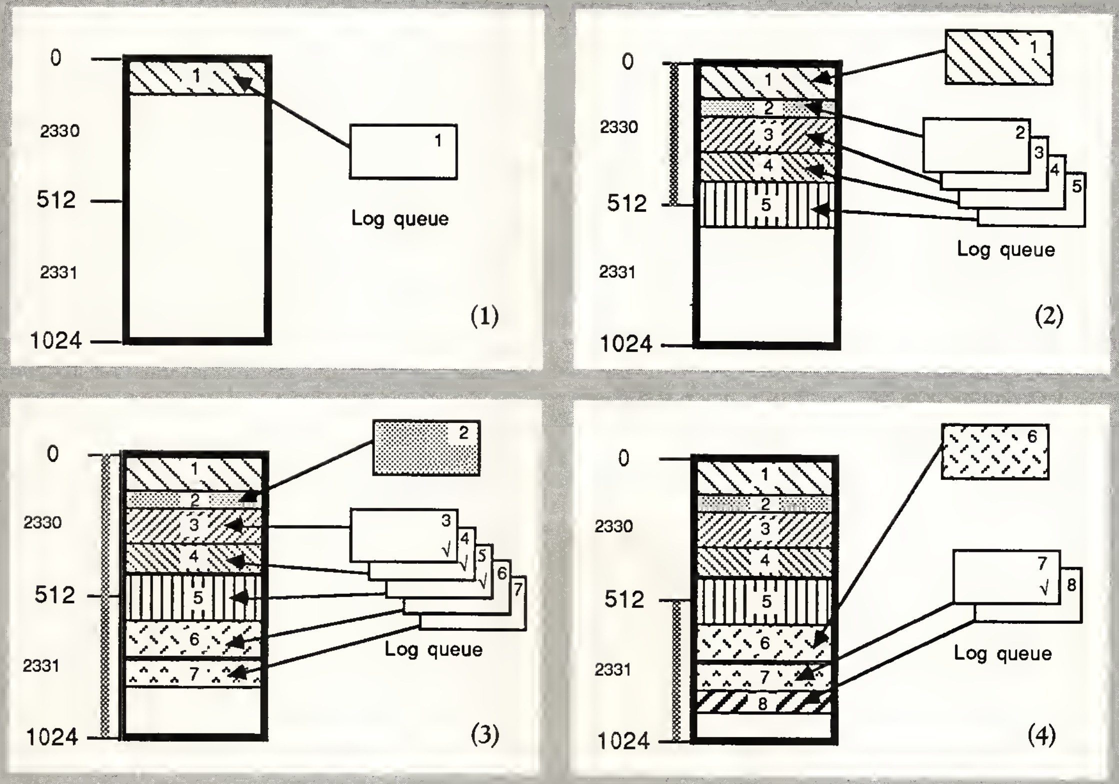


Figure C. Queue element purging by the log task.

downs will properly handle all pending or incomplete atomic transactions.

Should an application begin an atomic transaction and then determine that it is unable to complete it, it will issue an *abort network request code*, which instructs the central file server to purge all pending updates associated with the aborted transaction.

Backup/restore. Earlier implementations of the central file server required files within the PCIS database to be taken off line from the network for backup. Since prudent system administration requires frequent, regular backup, this meant frequent, regular downtime for at least some users of the network. Moreover, the network couldn't be used at all when certain key files were taken off line for backup. The current implementation permits on-line file backups, with all users retaining full utilization of the network while files are being dumped for backup.

A file may be updated while it is being dumped for backup. Therefore, after the dump has been completed, the backup copy may no longer reflect the true contents of the file. Compensation for this is made—with the aid of the critical event log, since all the updates to the database are recorded in it before they are applied to the database—if and when the file is restored.

The backup and restore program communicates with the central file server via the AOS/VS interprocess communications facility. To start the backup of a file, the dump process sends a request to the central file server specifying the file to be dumped. The central file server writes a start-of-dump record on the critical event log and returns the position of this record in the log file. This position is in turn recorded in a header record on the dump tape. Restoring a file includes creating a disk file from the tape copy, finding the position in the log file at which the dump began, extracting from the log all updates to the file being restored from that point forward, and applying these updates to the file. The file can be restored up to the time of the latest logged event, or up to some specific time before that event, as specified by the system administrator.

Although one must take a file off line to restore it, one can complete this operation with the network active and all other files of the database on line. Thus, any adverse user impact is limited to those applications that must access the file being restored.

The backup and restore program maintains a catalog of up to 10 generations in each of three categories of backup tapes for each file of the database. The system administrator need specify only the name of the file to be restored; the program then provides the volume serial numbers of all the tapes required to effect the restoration. These tapes include the tape (or tapes) containing the most recent backup of the file to be restored as well as the tapes containing the required segments of the critical event log. (The critical event log is logically segmented on disk; when a segment fills, it is dumped to tape.) The program's automatic maintenance of the catalog and quick retrieval of information from it have a definite positive effect on recoverability. These services reduce the workload of the system administrator, shorten the time required for the preparation of a restore, and lessen the probability that human error will impede the restore process.

Information contributes to availability. In a complex system, human intervention can contribute to availability. The contribution can have positive or negative influence, depending on the timing and nature of the intervention. Often the difference is determined by the individual's understanding of system behavior, the information available to him for making judgments, and the tools available to him for analyzing that information.

The current implementation of the central file server includes several tools and information aids designed to help the system administrator obtain maximum performance and availability from the system. Some of these tools have already been mentioned. Others are described briefly below.

Programs to help define the system configuration prior to installation are provided. These programs check for and report any inconsistencies in the configuration and report the memory, file space, and other resources the configuration needs. Tools are provided to help determine, during installation, the optimum location of database files on the disk drives and to compute the optimum file element size (a performance vs. space trade-off) for each file. These tools can also be used for the same purposes when changes to an existing configuration are required.

Each time the central file server is activated, checks are made to determine whether configuration changes have been made and, if so, whether those changes are consistent. Inconsistencies will usually prevent the central file server from activating the network.

A formalized method for reporting and logging exceptional conditions is integrated into the central file server and is used by any routine that encounters such a condition. Each encounter is reported to the system console device and written to a disk-resident log. Each message is time- and date-stamped, and each briefly describes the nature of the condition encountered. Also displayed with each message are a severity code and a message number. The message number serves as an index into a manual which contains a detailed description of the condition that produced the message and possible courses of action. A program for perusing the exceptional condition log file is provided, as is one for producing summary reports from its contents.

Although the critical events log was primarily implemented to permit automatic recovery from unscheduled shutdowns, it is also an excellent source of information about system behavior. A tool for browsing the log file, and for producing summary reports of the events recorded there, is provided. Reports can be produced by time of day, by workstation, by workstation user id, by database file, or by type of event. By combining these reports, one can obtain detailed views of system behavior.

Current-generation 32-bit super-minicomputers, with their large memory address space, high performance, and reliable operation, provide an excellent base for the implementation of features in a network's central file server that are designed to make that network highly available. Furthermore, 32-bit machines have enough extra power over that of their 16-bit predecessors that they can handle the overhead associated with high availability and yet provide higher performance. For example, a maximum disk I/O rate of about 20 disk blocks per second has been estimated for the 16-bit implementation (no facilities were incorporated in the 16-bit system for actually measuring this parameter), whereas sustained effective disk I/O rates (including cache hits) as high as 664 blocks per second have been measured on the 32-bit implementation.

We considered the features we installed to enhance system availability essential, even though they frequently had a negative effect on performance. We compensated for this effect by doing additional work on those aspects of the features that affected performance and by adding other features designed solely to improve performance.

The inclusion of facilities for on-line backup of files eliminated a major source of scheduled downtime. And the designing and coding of performance-enhancing features into programs that perform recovery functions further reduced downtime.

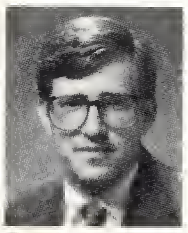
Availability and performance were enhanced by providing users with tools for obtaining a good system configuration and for monitoring and analyzing events that reflect system behavior. The automation of some recovery procedures and the automatic consistency checking of configuration changes increased system availability by reducing the likelihood of downtime caused by human error.

References

1. L. Svobodova, "File Servers for Network-based Distributed Systems," *Computing Surveys*, Vol. 16, No. 4, Dec. 1984, pp. 353-398.
2. W. Stallings, "Local Networks," *Computing Surveys*, Vol. 16, No. 1, Mar. 1984, pp. 3-41.
3. C. J. Date, *An Introduction to Database Systems, Vol. II* (*Addison-Wesley Systems Programming Series*), Addison-Wesley Pub. Co., Reading, Mass., 1983, pp. 1-33.
4. W. Kim, "Highly Available Systems for Database Applications," *Computing Surveys*, Vol. 16, No. 1, Mar. 1984, pp. 71-98.
5. M. B. Friedman, "Performance and Tuning of Cached I/O Subsystems," *Proc. CMG XV Int'l Conf. on the Management and Performance Evaluation of Computer Systems*, Oct. 1984, Computer Measuring Group, Phoenix, Ariz., pp. 717-727.
6. K. H. Bates and M. TeGrotenhuis, "Shadowing Boosts System Reliability," *Computer Design*, Vol. 24, No. 4, Apr. 1985, pp. 129-137.
7. W. Wilkinson and M. Lai, "Managing Replicate Data in JASMIN," *Proc. Fourth Symp. on Reliability in Distributed Software and Database Systems*, Oct. 1984, IEEE Computer Society Press, Washington, DC, pp. 54-60.
8. P. Ciompi, L. Simoncino, M. La Manna, and C. Lissoni, "A Redundant Distributed System Supporting Atomic Transactions for Real-Time Control," *Proc. Real-Time Systems Symp.*, Dec. 1983, IEEE Computer Society Press, Washington, DC, pp. 258-267.
9. D. Reed, "Implementing Atomic Transactions on Decentralized Data," *ACM Trans. Computer Systems*, Vol. 1, No. 1, Feb. 1983, pp. 3-23.

William A. Jackson is a systems engineer with EDS Research. He is currently working on interfacing IBM mainframes to Ethernet LANs using fiber optics and other high-speed communications techniques. He is interested in microprocessors, operating systems, communications, microprocessor-based interfaces, and performance issues.

Jackson obtained a BSEE degree from the University of Arkansas in 1968. He is a member of IEEE, ACM, and NSPE.

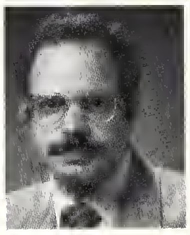


Paul C. Rogers is team leader of the group responsible for continued enhancements to the Central File Server at Electronics Data Systems Corporation. He has concentrated on systems programming in the areas of microcomputers and minicomputers.

Rogers earned a BS in computer science in 1977 from Mississippi State University, where he also worked as an assistant to the Systems Programming staff. In 1983 he earned a master's degree in Semitics and Old Testament studies from Dallas Theological Seminary, where his research interests included digital image enhancement of ancient manuscripts and statistical linguistics.

Robert J. Hearn is a systems engineer with EDS Research, where he currently works on automated programming methods. He was with International Computers Limited for seven years before moving to the US. His interests include expert systems, programming methods, and programming languages.

Hearn obtained a BSc in mathematics and an MSc in numerical analysis and computing from the University of Manchester, England.



Jeffrey S. Mattiace is currently concentrating on enhancing the Central File Server with increased automation in the backup and restore functions. He has been with the Health Services Divison of EDS since 1982, designing and implementing several PCIS subsystems, including communications interfaces to IBM and DEC systems, a parameter-driven DBMS, and applications development utilities.

Mattiace holds a certificate in computer programming from Airco Technical Institute.

Jackson and Hearn can be contacted at EDS Research, 7223 Forest Lane, Dallas, TX 75230; Rogers and Mattiace are located at the Electronic Data Systems Corporation, Health Services Division (A3-3E-03), 5400 Carpenter Road, Plano, TX 75024.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 162 Medium 163 Low 164

An Introduction to the V-System

Eric J. Berglund
Stanford University

A distributed operating system runs on Sun and VAXstation-II workstations connected by an Ethernet. Its distributed kernel provides inexpensive processes and fast interprocess communication.

Over the past three years members of the Distributed Systems Group at Stanford University have published a number of papers and delivered several talks describing experimental research in distributed systems. Most of this research has been conducted on the facilities of the V-System, a distributed operating system that relies on inexpensive processes and efficient interprocess communication to provide most system services at the application level. The research reports have provided a general description of the V-System, but they have gone into only as much detail as readers and listeners needed to understand the reported results. Here, I describe the V-System more extensively (though informally) to give the reader a better "feel" for its design.

The V-System is not just a testbed for research. Members of our research group and of other groups use it for their daily computing: electronic mail, document preparation, game playing, and so on. Thus, the decisions we make regarding the architecture of the system are not necessarily research-driven; they are also determined by what is practical and enjoyable for us to use. Because every group of workers building a practical system faces the same problems and solves them in some way, the solutions don't get much attention in the literature. Yet we feel there is some need for discussion of the problems designers of distributed systems encounter and the solutions they arrive at. We feel there is a need to gain insight into the engineering process involved.

System principles and goals

The V-System, like many of today's distributed systems,[1-6] is based on the client/server model.[7] In this model, system resources are managed by *server* processes, which implement policies to ensure desired levels of sharing, fairness, efficiency, and data abstraction. When a user or a process wants to use a resource controlled by such a server, it sends a request, thus becoming a *client* of that server (Figure 1). A server defines the abstract representation of its resource(s) and the operations on this representation. A resource may be accessed or manipulated only through its server. Because

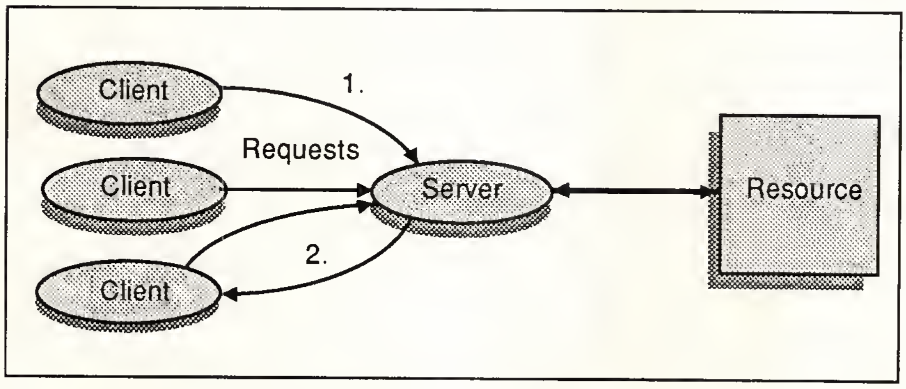


Figure 1. The client/server model.

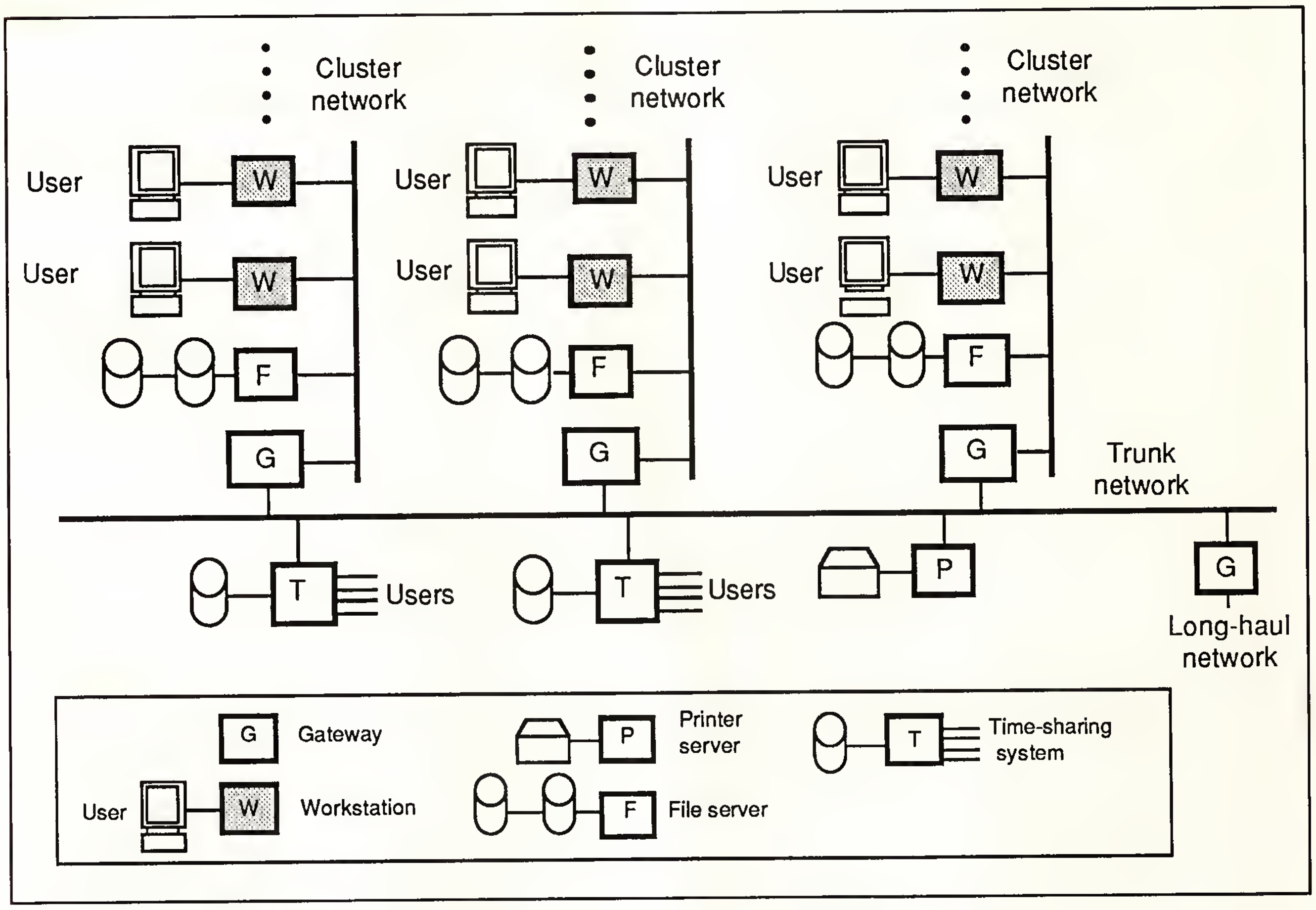


Figure 2. A workstation-based distributed system.

servers are constructed with well-defined interfaces, the implementation details of a resource are of concern only to its server. Of course, one server can become a client of another when it needs to use the other's resources. Note that the client/server model requires some mechanism that processes can use to send requests and replies to one another.

Clients and servers may be distributed throughout the (inter)network. By default, access to resources is *network-transparent*: A client may access a remote resource with the same semantics as those it uses to access a local resource. The result is an environment in which clients may communicate with servers without regard for the topology of the distributed system as a whole.

Within this framework, the V-System design is guided by the purposes we set out for it. As in many such efforts, the principal players in the Distributed Systems Group have particular religious convictions they want to preach. We believe and want to prove

• that a powerful system can be built upon primitives that provide inexpensive process management and simple, fast interprocess communication (if one makes these primitives extremely efficient, one can provide the high-level functions needed by some applications without burdening others with unnecessary overhead),

• that for communication between two processes, synchronous message passing (in which the sender of data waits until the data are received before continuing execution) provides the simple interface demanded above and can be implemented with the desired efficiency,

• that many common systems problems can be solved elegantly and efficiently with primitives that provide communication to and from *groups* of processes,

• that powerful, customizable user interfaces that support graphics and allow the user to perform more than one task at a time can be provided without significantly degrading performance,

• that a uniform interface and protocol, reasonably independent of particular physical devices or intervening networks, can be defined to simplify the process of adding new services to the system, and

• that principles learned in constructing distributed operating systems will be applicable to the design of operating systems for multiprocessor machines.

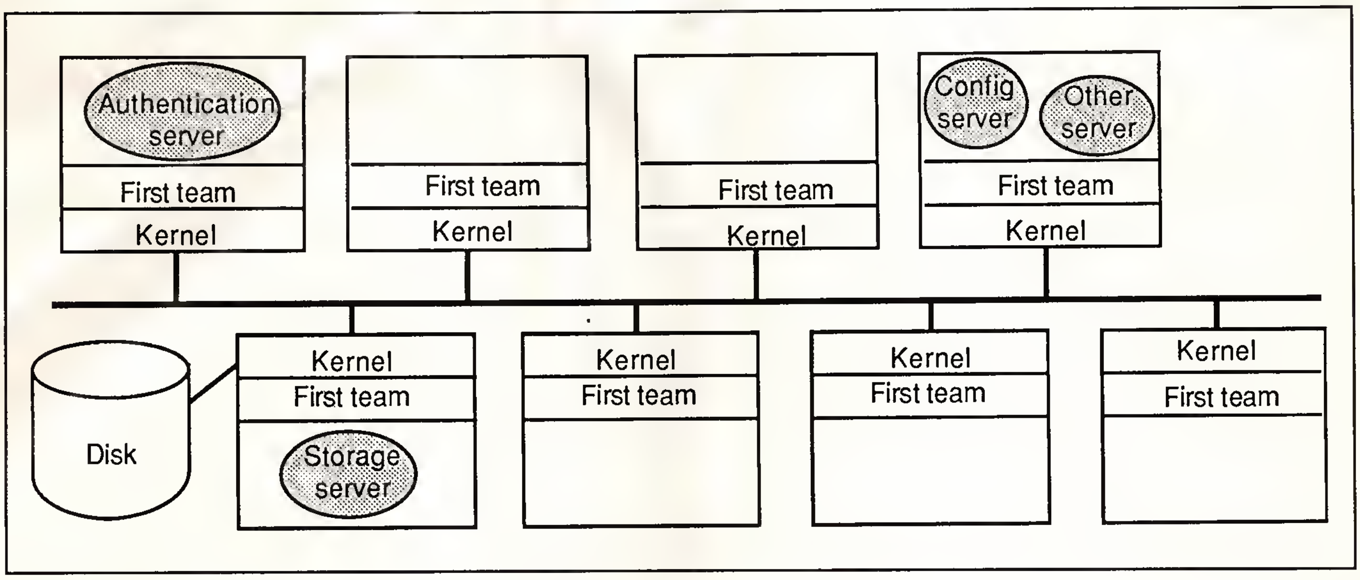


Figure 3. The V-System—a local-area network view.

Besides its "Shroud of Turin" status, there are other reasons for building the V-System. We hope to provide an environment for developing and studying distributed applications, including extremely robust systems. We hope to determine the hardware and software facilities needed to build efficient, autonomous, reliable, and available tools. We hope to make these facilities easy to use for the novice and powerful enough for the experienced user. We hope that our solutions will be portable to a variety of processor and network configurations. And finally, we share a wider faith with all proponents of distributed systems: Once such systems have achieved their potential, we believe they will be faster, cheaper, and more reliable than existing machines.

The hardware environment

The V-System is targeted for a hardware environment consisting of

• powerful, single-user workstations having a high-resolution (1024 × 1024) raster display, a general-purpose 1-MIPS or better processor, 1M byte or more of local memory, a large (greater than 20-bit) virtual address space, a graphics input device such as a mouse, and an optional disk;

• a fast (greater than 1M-bit/s) communications network linking the workstations;

• a number of dedicated processors providing printing, file storage, general computation support, and other services; and

• access to time-sharing or special-purpose computers and to long-haul computer networks.

The hardware environment is shown in Figure 2.

V-System 6.0 runs on Sun-1,[8] Sun-2, Sun-3, and VAXstation-II workstations interconnected by either a 3M-bit or a 10M-bit Ethernet.[9,10] A "guest-level" implementation is available for 4.2 BSD Unix systems. V also supports dedicated server machines providing file storage and printing, among other facilities.

The V-System kernel

The heart of the V-System is the *V kernel*,[11-13] a distributed kernel that implements a program environment of many lightweight processes communicating by messages. A copy of the kernel code runs on each machine in the system, providing the processes and interprocess communication upon which all other system functions are built (Figure 3).

Kernel operations can be divided into three categories: process and memory management, interprocess communication, and device management.

Process and memory management. A process in V is a logical activity that sequentially executes instructions. Associated with each process is a priority, a state, a team space, and a stack (Figure 4). The process's priority dictates the preference given to the process during processor allocation. The highest priority process that is *ready* is allocated the processor. The state is essentially the machine state of the processor for that process. The team space is the area of memory to which the process has direct access; there can be more than one process in a team space. The process's stack is a local memory area contained in the team space that the process uses for local workspace, procedure linkage and return, and the like. All processes with the same team space are said to be on the same team.

Processes can be dynamically created and destroyed by the sending of a message to the *kernel server* process inside the kernel. When a process is created, it is created as part of

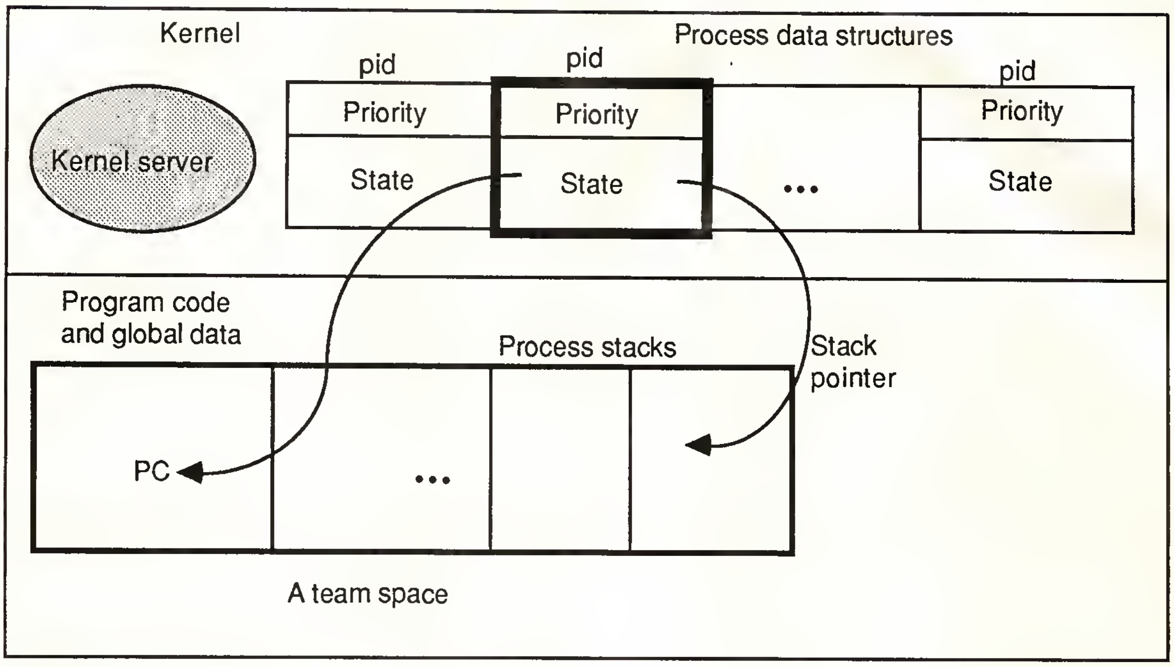


Figure 4. A V-System process.

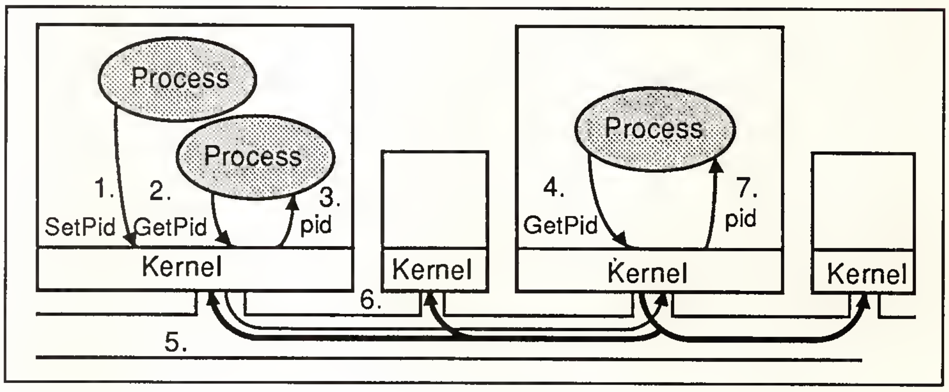


Figure 5. Kernel naming primitives.

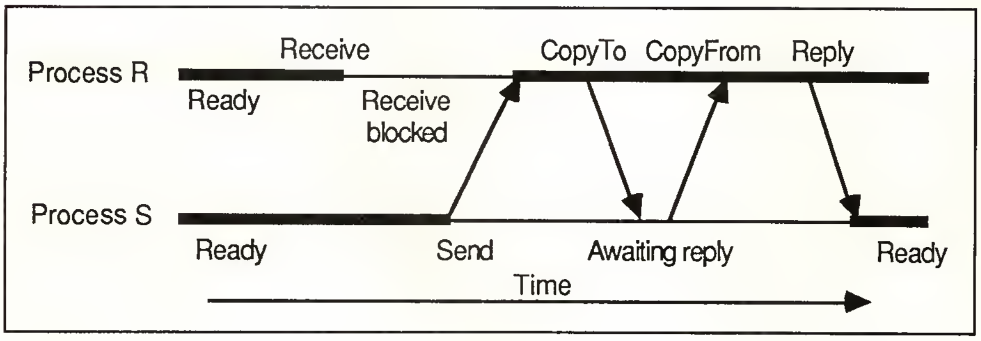


Figure 6. Synchronous message passing.

the same team as its creator's and is assigned a unique process identifier, or *pid*. When a process is destroyed, all the processes created by it are also destroyed.

A process can register its process identifier as corresponding to a particular *logical process identifier*, using the **SetPid** primitive (see Step 1 in Figure 5). A logical process identifier is merely a well-known integer reserved for a specific kind of server. Other processes can then query the kernel with the well-known integer, using **GetPid**, to learn the real process identifier (Steps 2 and 3). (Indeed, this is how a process can find out the pid of the kernel server mentioned above.) Registration of the logical-to-real process identifier can be specified as local to a workstation, if this is desired. If a remote process asks for the process identifier (Step 4), the request is sent to all of the kernel servers on the local network (Step 5), and the response comes from the kernel server on the appropriate machine (Steps 6 and 7).

The kernel uses the team space as the unit of memory management; the team space is always contained on a single workstation. Operations are provided for creating a process in a new team space, and for querying and setting the size of a team space. A team space disappears when the last process contained in that space is destroyed, so there is no explicit operation for destroying a team space. Teams are intended to provide fine-grain sharing of code and inexpensive sharing of data between cooperating processes. They separate the ideas of program, executable unit, and address space from that of process.

Interprocess communication. The kernel provides three forms of interprocess communication. First, processes may send, receive, reply to, and forward fixed-length messages. A process that has sent a message is suspended *awaiting reply* until the message it sent has been received and replied to by the receiving process. A process that is awaiting reply from a process that has been destroyed since the sending of the message is unblocked by means of an indication that the receiver of the message no longer exists. (A watchdog process, which is responsible for detecting the termination of a process, can take advantage of this "nonexistence notification" by waiting for a message—one that is never intended to be sent—from a process it is assigned to watch. Thus, it remains blocked until the process is destroyed.) In the current implementation of the V-System, messages are eight full words, where a full word is defined to be the maximum of the space required for a general machine pointer and the space required for a "natural" machine precision integer (32 bits on the MC68000-based Sun workstation and on the VAXstation-II).

Second, a process can pass access to a single segment (a contiguous range of addresses) in its team space to the recipient of its message. The recipient process can access this segment for reading and/or writing, depending on the access specified by the sender, while the sender is awaiting reply from the recipient. The presence of a segment and its access modes are specified in the message; its location and size are also given.

Note carefully the power the message sender delegates to the receiver. The sender can give access to its entire team space to the receiver, if it chooses to do so. The kernel provides the primitives **CopyTo** and **CopyFrom** to allow the receiver to read and write in the specified segment in the sender's address space while the sender is blocked. The receiver can execute these primitives as many times as it wishes to before replying (see Figure 6). Even if the sender permits no access to a segment of its address space, it has just put its own execution future in the hands of the receiver: The sender is blocked until the receiver deigns to reply. The receiver is free to *forward* the message to any other process it chooses, and when it does so, it passes not only the message but also the segment access and the right to reply. The team server/executive structure described later in this article depends heavily on this characteristic of synchronous message passing.

Synchronous message passing can be used to implement interprocess communication that looks to the sender like procedure calls. Just as a procedure caller cedes control to the procedure, so our sender yields to the receiver. The **Send** request message passes the equivalent of procedure arguments, and the reply message returns the results. Because the receiver can choose when it wants to reply to each message, it can receive and queue as many messages as it chooses, thus allowing sophisticated scheduling of message handling and replies.

The segment access operations follow the procedure paradigm as well in that they are used primarily to access what are logically "call-by-reference" parameters. The arguments for providing exactly one segment are that at least one is needed and that one is sufficient for the dominant activity, namely, file access. It is expensive and difficult to provide an arbitrary number of segments—having just one allows a simpler and more efficient implementation.

The third type of interprocess communication supplements one-to-one messages with a one-to-many (one-to-a-group) communication facility.[14,15] We define a *process group* as a set of one or more processes identified by a *group id*; the processes may be on different machines. All processes in a group are equal; no one process is distinguished from any other. Processes can freely join or leave groups and are free to join multiple groups.

Any process, including one that is not a member of a group, can send to a group. Sending to a group involves specifying a group id instead of a process id as the identifier to **Send**. Group ids are identical in format to process ids. A process may elect to receive 0, 1, or more than one reply message when it sends to a group. The 0-reply case is handled as an unreliable multicast to the group; the sending process does not block. The 1-reply case blocks the sender until it receives one reply message; this case assures reliable delivery to at least one process. Further replies are discarded and no indication of how many other processes received the message or replied to it is given. This form of communication is the same as one-to-one reliable interprocess communication to the first respondent and unreliable datagram communication to the rest of the group.

The multiple-reply case is similar to the 1-reply case except that in this case the second and any subsequent reply messages are queued in the kernel for the sender to retrieve until the start of the next message transaction, i.e., the next **Send**. Subsequent replies are received with the **GetReply** primitive (Figure 7). It is left to the sending process to decide how many replies it wishes to receive and how long it is willing to wait for them.

Device management. Devices managed by the kernel are accessed through the device server pseudoprocess inside the

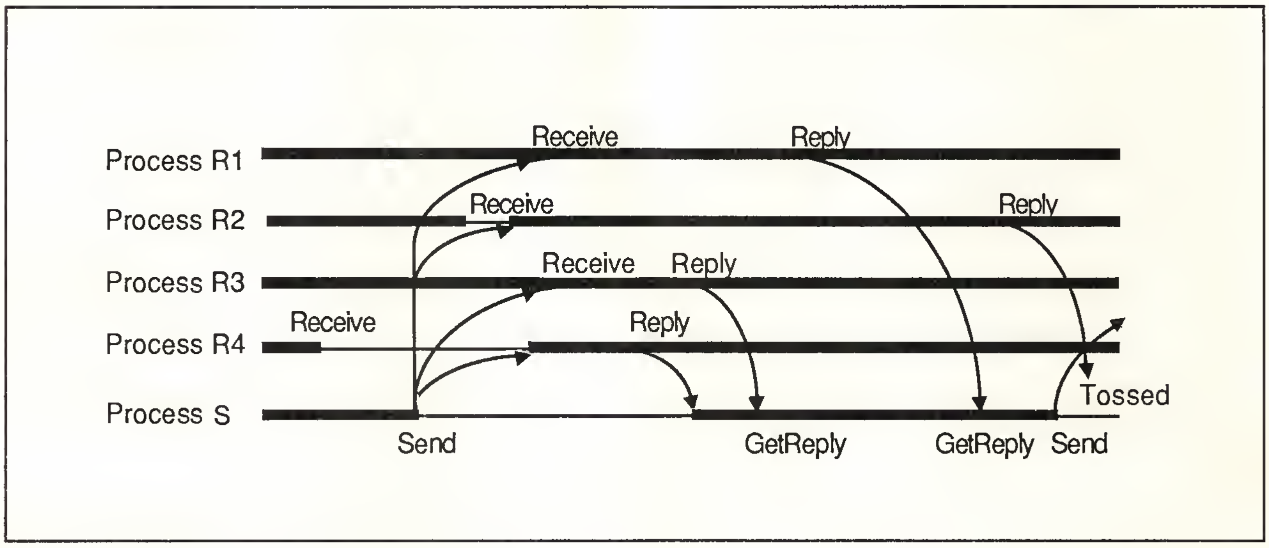


Figure 7. Group interprocess communication.

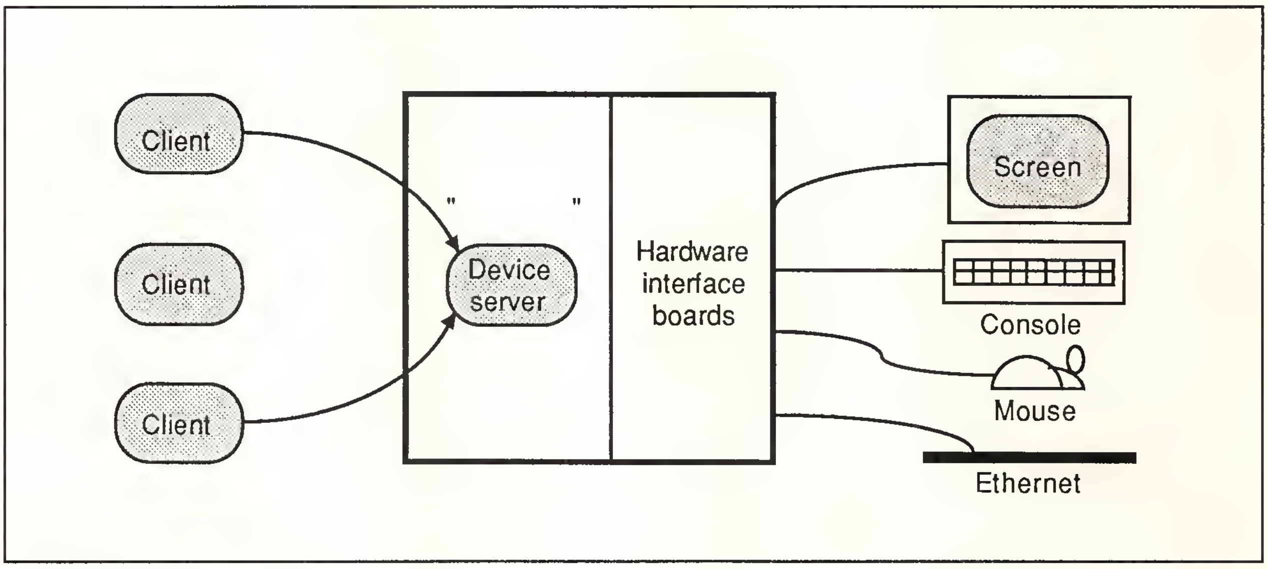


Figure 8. The device server.

kernel. These devices usually include the console, a mouse, and the Ethernet interface (Figure 8). Any process can learn the pid of the device server by using the **GetPid** primitive mentioned above. It can then ask for operations to be performed by sending messages to the device server, using the V-System's I/O protocol.[16] The I/O protocol is designed to provide uniform transfer of data between a client and a server managing the data as a *file*. The file is a general concept referring to many different types of objects that can be viewed as readable or writable. The protocol provides operations for opening, reading, writing, querying, and closing *file instances*. (This approach has proven to be of surprising generality. Among the objects that are managed by servers implementing the I/O protocol are screen windows, the keyboard, the mouse, internetwork connections, pipes, and traditional files.) Devices that can be controlled without special kernel support can be handled directly by processes. Special devices that require kernel support but do not fit the I/O model assumed by the protocol can be handled by the addition of new kernel operations.

A distributed application

To test many of the kernel primitives, we developed a distributed game called Amaze[17]; its structure is illustrative of that of many of our multiprocess programs. A game of Amaze is played with two to five players; each player uses a separate workstation running a copy of the Amaze program. The workstation display shows a maze and the "monsters" of all the players (Figure 9). The object of the game is to move the monster around the maze and shoot the other players' monsters.

Each copy of Amaze maintains a complete copy of the game state and updates the state and its display according to its player's instructions, the passage of time, and the actions of other players. When one player moves or shoots, an update message describing his action is sent to the other workstations in the game. Thus, each copy of the Amaze program must be able to handle updates from each of the other players, keyboard input from its own player, and real-time animation. To enable it to handle these multiple concurrent events, we structured Amaze as a set of multiple cooperating processes; in doing so, we followed the principles of *multiprocess structuring* by means of request-response message passing, which were developed in Thoth.[18]

We use one process to represent each asynchronous activity in the application. Thus, each copy of Amaze has a keyboard reader process, one *status inquirer* process to monitor each of the other players' monsters, and a timer process to notify it when the animated display should be updated. Each of these processes sends a message to the *game manager* process, which maintains the local copy of the game state. Figure 10 depicts, for a three-player game, the set of processes and the messages sent by the players.

Since the game manager is the only process that updates the state, it effectively serializes the update messages. After initializing the game state and creating its *helper processes*, the game manager enters an endless loop in which it waits to receive a message from one of its helpers reporting that

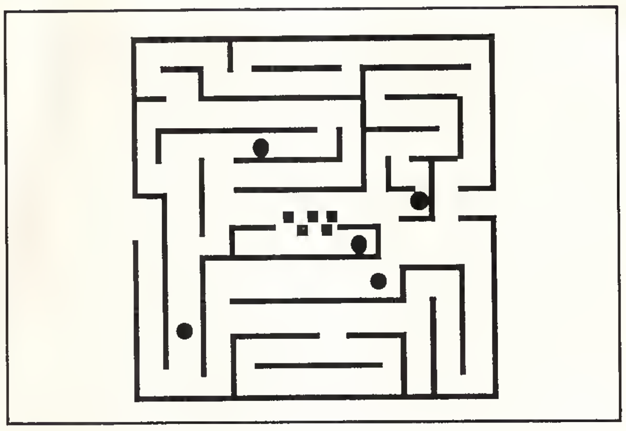
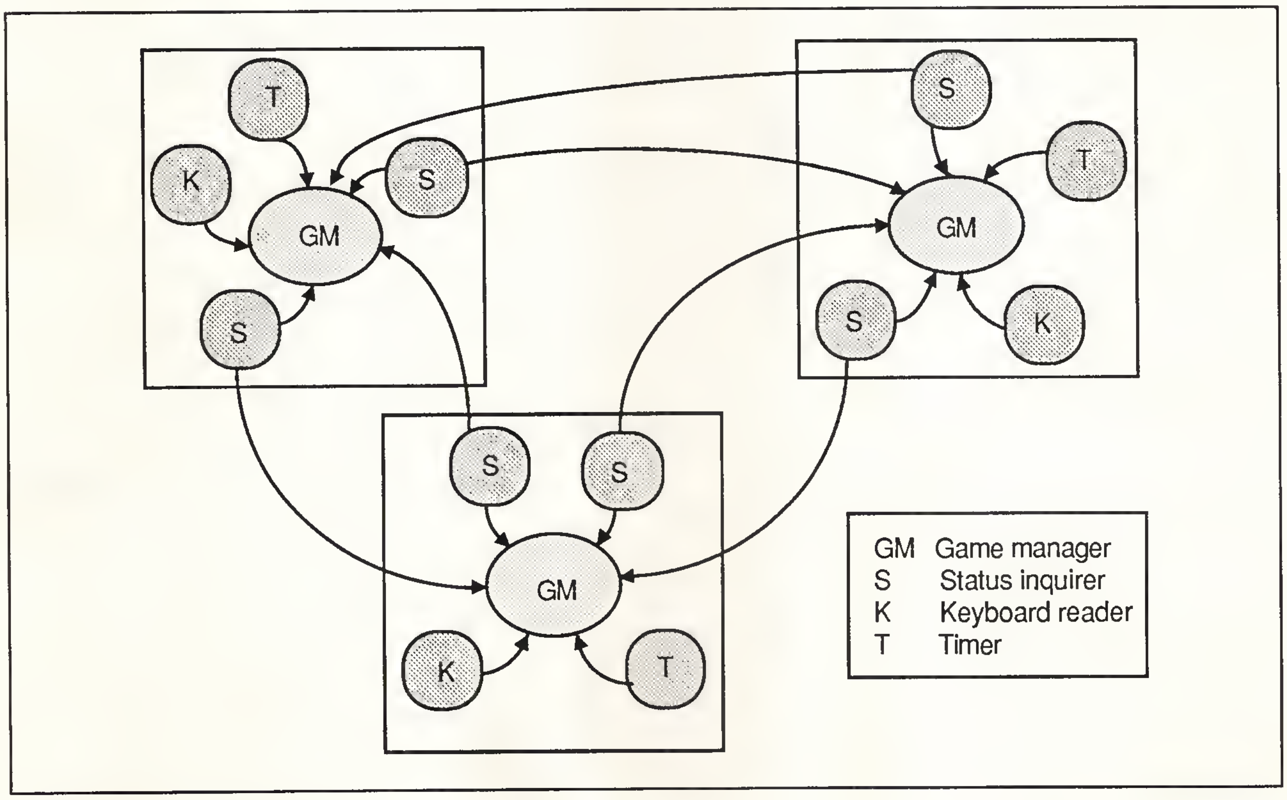


Figure 9. An Amaze display.



GM	Game manager
S	Status inquirer
K	Keyboard reader
T	Timer

Figure 10. Amaze processes.

something has happened. The simplest helper processes merely wait for some event to occur and send a message to the manager when it does. Two examples are the GameTimer and KeyboardReader processes (Table 1).

Note that the helper processes are told the pid of the game manager so they can send messages to it. The pid serves as a loose form of *capability* or "ticket." Possession of a pid is sufficient to allow any process to send a message to another process. When a helper process sends a message to the game manager, the manager takes the appropriate action, updates the game state, and replies to the helper process, freeing it to watch for the same event again. The heart of the game manager code looks like that in Table 2.

V-System processes are sufficiently inexpensive to permit the use of many small processes to achieve the desired level of concurrency in an application. Many of our applications and almost all of our servers are structured as Amaze is: One manager process receives all requests and synchronizes access to resources and common data structures. Smaller processes are created dynamically to monitor asynchronous activity and are destroyed when they are no longer needed.

Amaze also takes advantage of the naming primitives provided by the kernel. When a new player's copy of the game begins execution, it executes a **GetPid** to find out whether any other people are playing the game. If so, the **GetPid** returns the pid of one of their game managers, and the new player's copy sends a message to that game manager requesting permission to join the game. If there is room for another player, the reply will include the pids of all the game managers currently playing so the new entrant can communicate with all of them. Each new manager allowed to join calls **SetPid** to register itself as an Amaze player. Its kernel will then know to volunteer the manager's pid when the next **GetPid** is broadcast.

Recently we constructed a version of Amaze that uses the group communication facilities of the kernel. Each game manager joins a group made up of the other game managers when it enters the game. Instead of sending its state updates to each remote player individually, the manager sends them to the group, using the 0-reply case. Note that it need not wait for replies to ensure that the remote managers have received the message, since it will soon be sending them

Table 1.
Helper processes: GameTimer and KeyboardReader.

```
GameTimer( mymanagerpid, delayclicks )
        ProcessId mymanagerpid;         /* The pid of the game manager. */
        unsigned delayclicks;           /* Time to wait between animation updates. */

  {
    Message msg;
    GameRequest *req = (GameRequest *) msg;

    while( 1 )                          /* Loop forever. */
      {
        Delay( 0, delayclicks );        /* Wait for time to pass. */
        req→requestcode = GAMETIMER;
        Send( req, mymanagerpid );      /* Tell the manager it has, and block until the manager replies. */
      }
  }
KeyboardReader( mymanagerpid )
        ProcessId mymanagerpid;
  {
    Message msg;
    GameRequest *req = (GameRequest *) msg;

    while( 1 )                          /* Loop forever. */
      {
        req→inputchar = getchar( );     /* Wait for a character to be typed. */
        req→requestcode = GAMEINPUTCHAR;

        Send( req, mymanagerpid );      /* Tell the manager what it was, and block until replied to. */
      }
  }
```

another real-time update that will supersede the preceding update.

Indeed, Amaze could be rewritten to use group communication in place of the **GetPid** and **SetPid** primitives. If the Amaze managers were assigned their own well-known group id, as many of the more important servers have been, a new manager could send its "join game" request to that group id and wait for one reply. The distributed kernel would notify the new manager if there were no current members of the group, so the new manager would know it was starting a new game. In either case, it would join the group so it could respond to future requests. Many of the servers in the V-System that used to be located by means of **GetPid** have now been assigned such a well-known group id. Clients use them by sending to that group id.

The servers on each workstation

When a workstation is booted, its PROMs load a program which loads the V-System kernel and the *first team*. After the kernel has completed its internal initialization, it creates an initial team space and an initial process on this team, and starts this process. The process starts all the servers needed to run the system on the workstation: the exception server, the team server, the exec server, and some version of a workstation agent (see Figure 11). All but the last are processes on the first team, and thus they share an address space.

The structure of all the system servers is similar to that of Amaze. After some initialization of data structures, each creates helper processes if it needs them to wait for asynchronous events and then enters an infinite loop to **Receive** requests for service. Some of the requests may result in the creation of other processes on the same team as the server.

As the reader can doubtless gather from the picture of the system given above, many of the servers depend on each other's services. Some, however, are more self-contained than others and depend primarily on the kernel facilities. It is those we discuss first.

Workstation agents. Workstation agents are a generic class of server used in the V-System. A workstation agent mediates between the terminal hardware, the user, and the other programs, including servers, in the system. It is called upon by programs or other servers to receive input from devices controlled by the user, or to display output to the terminal screen. Workstation agents read from the keyboard and the mouse by sending requests to the kernel device management routines. (Of course, helper processes send these requests and then block until they receive a reply with new input. They then send a message to the main workstation agent process so that it can take appropriate action. The agent doesn't send the request to the kernel itself because it would then be blocked waiting for input and couldn't fulfill any output requests while it waited.)

Each workstation has one workstation agent running on it. All our workstation agents provide basic line editing capabilities. They make sure that the backspace key does not add a backspace character to the input stream but instead deletes the previous character, and they translate the new-line character into a carriage return/line feed sequence on terminals that require it. Workstation agents may, but

Table 2.
Game manager code.

```
if( (state = Initialize( )) = = NULL ) return;

while(1)
  {
   pid = Receive( req );          /* Receive the next message. */

   switch( req→requestcode )   /* Take appropriate action.*/
      {
       case GAMEINPUTCHAR:
           reply = ProcessInputChar( state, req, pid );
           break;

       case GAMETIMER:
           reply = ProcessTimer( state, req, pid );
           break;

       case REQUESTMONSTERSTATE:
           reply = ReturnLocalMonsterState( state, req, pid );
           break;

       case REPORTMONSTERSTATE:
           reply = UpdateRemoteMonsterState( state, req, pid );
           break;

       case JOINGAME:
           reply = JoinGame( state, req, pid );
           break;

       case AUTOPILOTCHAR:
           reply = ProcessAutoPilot( state, req, pid );
           break;

       case MISSILEREPORT:
           reply = NoteMissile( state, req, pid );
           break;

       default:
           reply = NOREPLY;
      }
   if( reply != NOREPLY )
      {
       replymsg→replycode = reply;
       Reply( replymsg, pid );  /* Reply to the process which */
                                /* sent the message, freeing */
                                /* it to wait for a new event. */
      }
  }
```

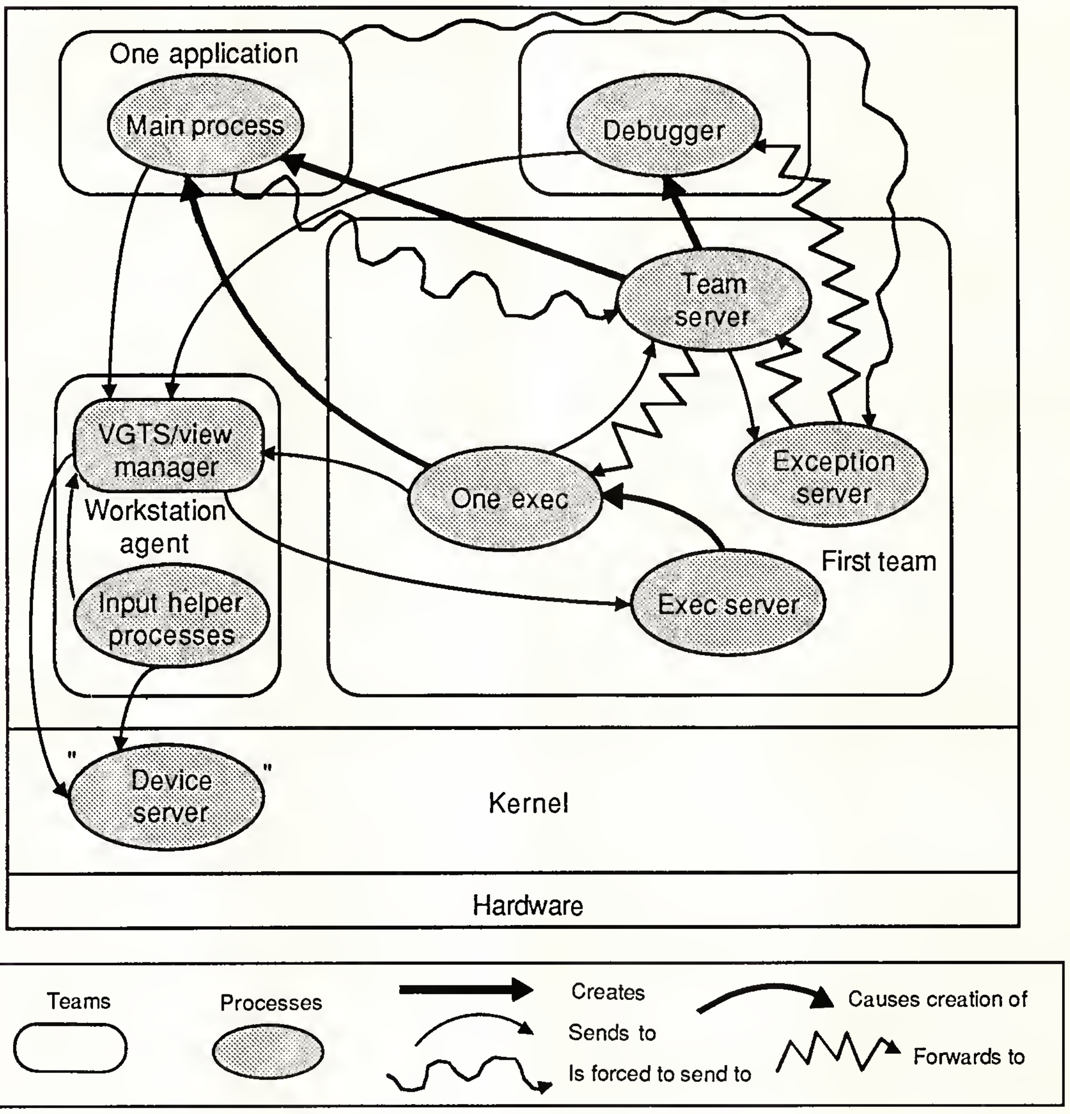


Figure 11. The V-System—a single workstation view.

need not, support multiple I/O streams. Our workstation agents can differ for two reasons: because they offer different services to users, or because they run on different kinds of terminals.

Simple terminal server. Currently the V-System can be configured with two different workstation agents, the simple terminal server, or STS, and the virtual graphics terminal server, or VGTS. The STS is a minimal workstation agent. It provides a single I/O stream, using the terminal facilities provided by the firmware monitor of the workstation. It gains access to these by sending messages to the kernel device server using the V-System's I/O protocol, which includes commands for initializing the device, querying its state, reading a "block" of data, and releasing access to the device. The STS implements a standard line editing facility but supports neither graphics nor a mouse.

The STS, besides taking requests from its clients for the next line of input, also handles requests for modifying the *input/output cooking*. For instance, it can turn off the feature that echoes character input, or it can set a switch that keeps a screenful of output from whizzing by before the user can read it. Indeed, though most programs can execute using any workstation agent, those that want low-level control of the screen must use the STS and ask it to turn off all the features it usually implements.

Virtual graphics terminal server. The VGTS[19,20,28] is our more powerful workstation agent; it provides a large set of facilities, including multiple I/O streams in multiple windows, graphics output, and mouse-controlled menus. It multiplexes both the output devices (the screen) and the input devices (the keyboard and mouse) among all the programs that use them. My screen looked something like

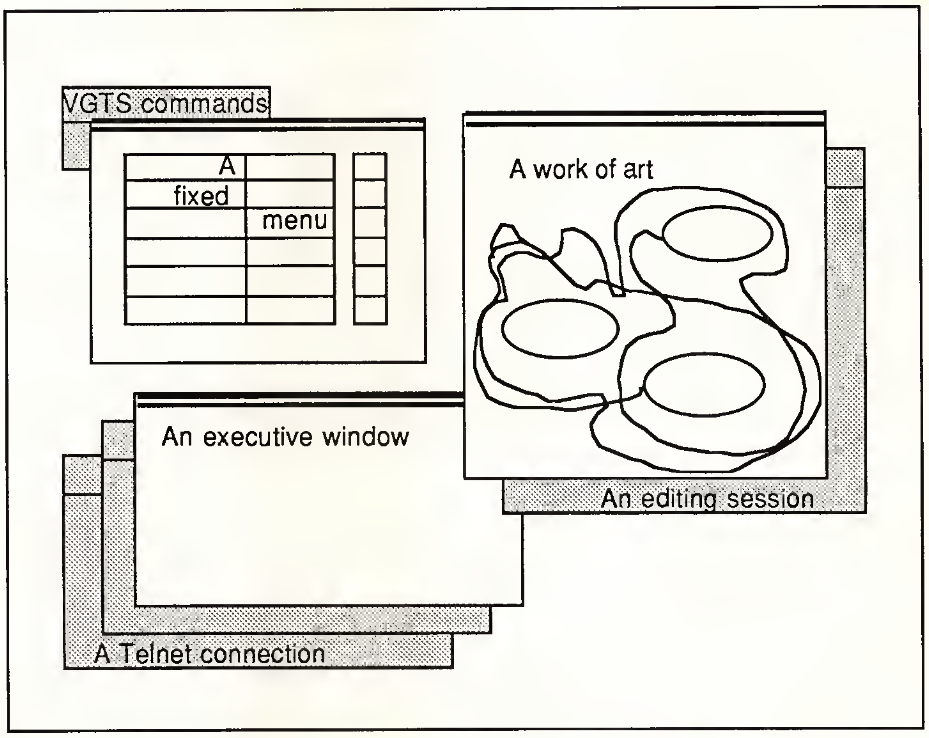


Figure 12. VGTS screen output.

Figure 12 when I was preparing this article. The command interface within each of the *executive windows* is identical to the command interface provided in only one window by the STS; includes the line editing and I/O cooking features found in the latter.

The VGTS is currently structured as one server process with three helper processes. There is one helper process to receive input from the mouse (again, through the kernel device server using the I/O protocol), one to read from the keyboard, and a timer process to invoke periodic functions like redrawing the screen. The mouse and keyboard reader processes are collectively called the *input handler* (Figure 13).

Clients of the VGTS can define and display *items*, which can be either graphical primitives such as lines, rectangles, text, and raster bitmaps, or *symbols* consisting of other items. They can also create virtual terminals to display items

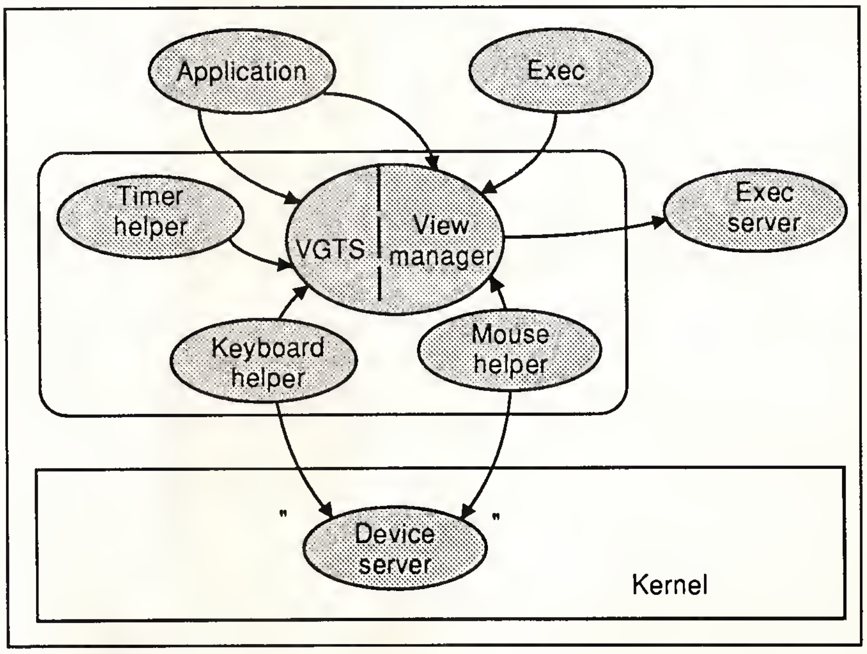


Figure 13. VGTS processes.

or to emulate ANSI-standard text terminals. The VGTS functions, like those of most servers, are accessible to applications programs through library routines. The library routines take appropriate parameters, format a message to the server, and send it. Thus, the message interface used for requesting system services is hidden from the casual programmer by a conventional procedure call interface.

Users can also use the facilities of the VGTS directly to control the layout of the screen and the way in which the virtual terminals are viewed. The module that performs this function is called the *view manager*. In fulfilling this duty, it acts logically as a client of the rest of the VGTS: It asks the input handling routines for mouse clicks, calls upon the drawing routines to display the appropriate menu, receives the next click for the command selection, and uses additional output routines to perform the selected operation.

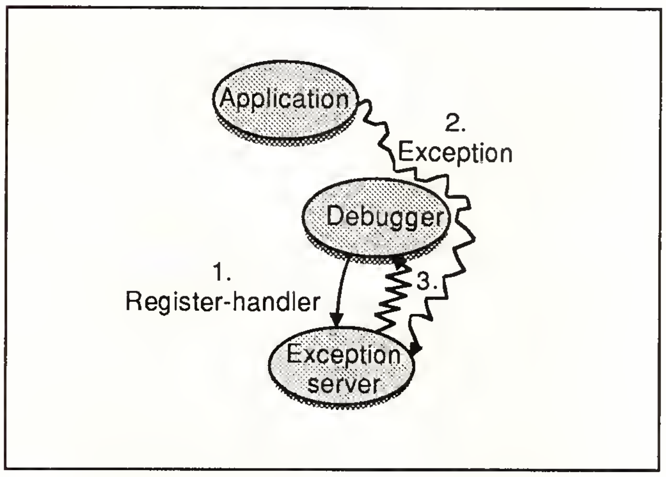


Figure 14. The exception server.

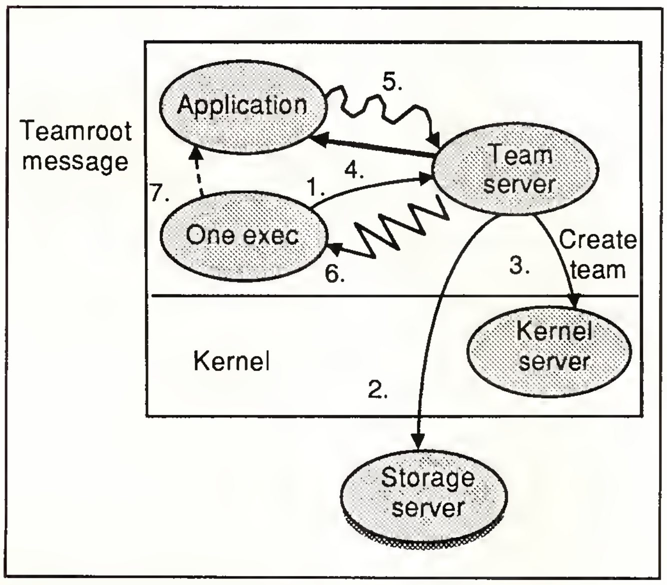


Figure 15. The team server.

The first team

Because users can replace a workstation agent by using the **newterm** command, the agent runs in a team by itself. Most of the servers running on each host, however, run on the *first team*, which is started by the kernel after initialization. The first team servers are the exception server, the team server, and the exec server.

The exception server. The exception server handles only two types of requests. The first is sent when a process incurs a processor exception during its execution. In this case, a trap occurs and is fielded by the kernel. The kernel effectively forces the process to send a message to the exception server, with the contents of the message describing the exception. The message gives the server complete access to the process's team space.

The exception server, upon receiving such a message, checks to see if another *exception handler* has registered interest in the exceptions caused by this process (Figure 14). For instance, if the program is being executed under control of the debugger, the debugger will have sent the exception server the other kind of request—a "register handler" request (Step 1 in Figure 14). The exception server will have recorded the pid of the process being run and the pid of the debugger that sent the request. Then, when the process incurs an exception—by hitting a breakpoint, for example—the exception message (Step 2) will be forwarded by the server to the debugger (Step 3).

If no process was registered, the exception server prints a message on the screen (using the terminal server) indicating the type of the exception and some state information, such as the value of the program counter. This case does not actually come up very often in practice because the *team server* registers itself as the handler of last resort for almost all processes.

The team server. The team server is the manager of the physical host. It loads, executes, and monitors all teams other than the first. (Recall that a team usually corresponds to a program, although some programs consist of more than one team.) Requests to the team server ask it to load and start a team, to terminate one, or to print the directory of currently executing teams.

A client of the team server (usually an exec, which I will describe later) must specify an open file descriptor from which the team can be loaded (see Step 1 in Figure 15); these files are usually managed by *storage servers* (Step 2). Teams can be loaded from object code files using routines in the V-System library. These routines package an appropriate request to the team server and take care of matters such as setting up the initial arguments on the stack of the team's initial, or *root*, process. The team server sends a message to the kernel server process (Step 3) requesting that a process be created on a new team. Like all processes, this process is created in the *awaiting reply* state (Step 4), waiting for a **Reply** from its creator. In effect, the kernel simulates a **Send** from the new process to the team server (Step 5). Recall the power this gives the team server as a receiver. In this case it gains access to the new team's entire address space as well as the right to **Reply** at any time, delaying the team's execution until it sees fit. The team server forwards this message and its associated privileges (Step 6) to the client that originally asked for the team to be loaded, called the team's *owner*. At this point, the client (or the library routine it calls) can put the arguments on the stack of the root process and then **Reply** (Step 7).

The reply message sent in this case is called the *team root message*. Among other things, it includes pointers to the I/O objects that the program will read and write, the environment variables, and the name cache.[21] These I/O objects are most often files or virtual terminals.

A function called **TeamRoot**, which is included in every program compiled for the V-System, reads the team root message, sets the team's standard input and output to the specified I/O objects, and calls **main**. If **main** returns, **TeamRoot** calls **exit**, which closes all of the team's open files and sends a request to its owner to terminate the team. If called with a parameter from within the team, **exit** includes the parameter in its termination request; its owner can use this *exit status* as it sees fit.

As described above, the team server uses only the services of the kernel and a storage server. However, the team server is also the principal client of the exception server; it registers itself as the exception handler (Step 1 in Figure 16) for all teams that don't otherwise have one. If a process on such a team incurs an exception, the exception message (Step 2), which the kernel forces the offending process to send, is forwarded to the team server (Step 3). The team server then uses its own facilities to load the V-System debugger (Step 4) and forwards the message to it (Step 5). Upon receipt of the message, the debugger prints the exception message, handles user commands, and registers itself as the exception handler for the team. The debugger may reply to the original exception message, freeing the team to continue execution until it either gets another exception or terminates normally. The next exception, if any, is handled by the debugger.

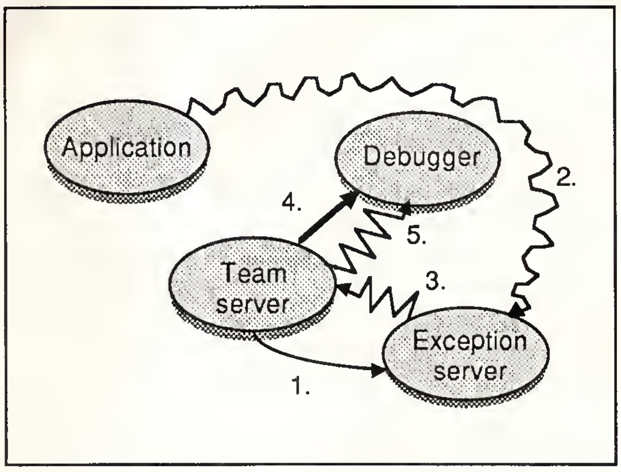


Figure 16. Exception handling.

The exec server and the executive. The principal client of the team server—and of the workstation agent—is the V-System *executive*, or *exec*. Where the workstation agents provide the graphical interface for the user, the exec provides the command interface. (See Lantz[22] for a discussion of the overall user interface architecture.) It corresponds to the Unix shell or the Tops-20 executive in that it is a user-level process providing command parsing and convenient access to system services.

The basic operation of the exec is to read command lines and execute commands. The first field on a command line is the command name; the rest are arguments to be passed to that command. A command name can be a built-in exec command, the name of a file containing a program compiled to run under the V-System, or the name of a program to be run on a storage server, such as the Unix server I will discuss below. The exec provides a simple search path mechanism for commands. If a version of the program that will run on the user's workstation cannot be found, the exec will try to execute the command remotely on a storage server.

The exec is a process which sends requests to the workstation agent—either the STS or the view manager of the VGTS—to read from the keyboard and write to the screen. The workstation agent provides line editing capabilities to the user. The exec itself provides command history facilities, aliases, I/O redirection, background commands, and a way to specify that commands be executed on another host running the V-System.

Execs are managed by the *exec server*, the last of the servers on the first team. The principal client of the exec server is the workstation agent. When either the STS or the VGTS is started, it creates a process whose sole function is to ensure that at least one exec exists at all times. This process sends a request to the exec server to start the first exec. The request has to specify the standard input and output instances the exec will use. For the STS, both I/O instances simply refer to the screen; for the view manager, they point to a virtual terminal that has been created for the use of the new exec.

One of the view manager's principal functions is to give the user a simple interface to the exec server. Mouse-controlled menus supplied by the view manager enable the user to send requests that call for the creation and deletion of execs or for the interruption or termination of a command running in an exec. Of course, the STS does not support all of these functions; since it provides only one I/O stream, it can allow only one exec to exist at a time.

Each exec is created as a process on the first team, and it sends requests to the server providing its standard input for one line of input at a time. It parses this line of input to determine the program to be executed and sends requests to storage servers (Step 1 in Figure 17) to open the file that contains the program and other appropriate files if input and/or output has been redirected. The exec then constructs the team root message for the program-to-be, using the server pids and file identifiers returned by the opening routines. Note that if I/O has not been redirected, the standard input and output for the program will be the same as those of the exec—the virtual terminal the exec was given for its own I/O. Having done all this, the exec sends a request to the team server asking it to load the new team; this request makes the exec the principal client of the team server (Step 2 in Figure 17).

As the reader will recall, the team server creates the new team (Step 3), leaving its main process awaiting reply from its creator (Step 4). The right to reply is then forwarded by the team server to the process that sent it the message—in this case, the exec (Step 5). Thus, the exec is given access to the team space of the new team, which it uses to write the new program's arguments on its stack in its own team space. After doing so, the exec replies to the program, an action which starts it (Step 6). Unless the program is being run in the background, the exec waits for it to complete execution by waiting for a message from it. (Programs running in the background have the team server as their *owner*. If the exec that started them is deleted by the user—which usually results in the destruction of any program the exec owns—the background program will continue execution.) Since a V-System program sends its exit status to its owner, the exec stays blocked until it receives the exit status or until the program terminates abnormally. At this point, the exec is ready to ask for the next command line.

Summary. The workstation agents and the first team servers combine with the kernel to provide all the system services on each single-user diskless workstation. One of the principles guiding the V-System design has been to put as many of the usual operating systems functions at the user level as efficiency permits. Another has been to give the sophisticated user an opportunity to design her own user interface. The design of the workstation agent, executive, and team server have attempted to keep these goals foremost. Thus, the ambitious user can build her own workstation agent providing the graphical interface of her dreams. She can replace the standard executive with one of her own construction, yielding a different textual interface. She can do both of these without rewriting the "hidden servers" that provide team loading, name service, and exception handling.[22]

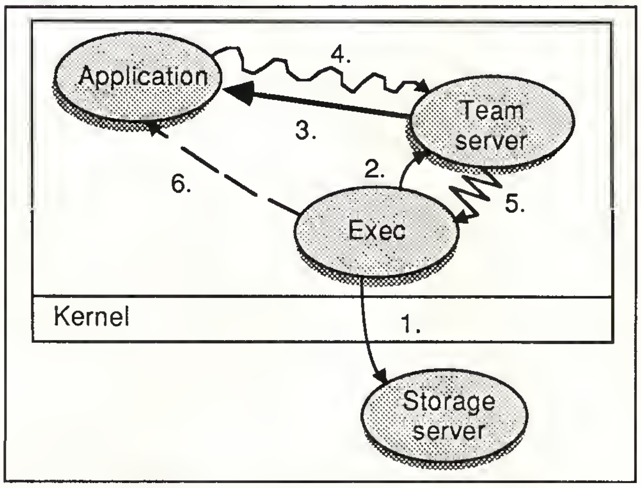


Figure 17. Starting a program.

System-wide services

One of the key arguments advanced for favoring distributed systems over personal computers is that in distributed systems services too expensive or too inconvenient to provide to a single user can be shared by many users. Not all the V-System servers run on each workstation; for some a single instantiation running on the local network suffices. The printer server, for instance, runs only on the workstation dedicated to spooling files and driving the printer. The two most important system-wide services are the storage server and the authentication server.

The storage server. The storage server[26] is a file system that implements the V-System's I/O protocol. It currently runs on a dedicated server machine (one that runs only servers) with mass disk storage; the kernel running on the machine must include support for the disk in its device server routines. The storage server can, however, run on any machine with a disk. Most of the machines running the V-System do not have disks, the expense and noise having been deemed excessive. Thus, the present storage server(s) must provide file access for users on the network.

Like many servers, the storage server is a member of a group of similar servers who share a well-known group id. A client can discover a particular server's pid using this group id and the V-System naming protocol.[21] Once it does so, it can create or open a file by sending the server a CREATE_INSTANCE request (Step 1 in Figure 18); a client usually does this by calling the **Open** library routine.*

The reply to this request (Step 5) includes the pid of a *file instance server*, a process created by the main storage server process (Steps 2, 3, and 4) to handle all subsequent requests dealing with the file. Thus, the scheduling mechanisms built into the kernel are in fact scheduling file access by schedul-

*In the V-System I/O protocol, a *file instance* is an object that is created and possibly initialized to contain the same data as an existing, permanent file. When the file instance is released by the client, the data contained in it are atomically written back to the corresponding permanent file. However, for some servers—the internetwork server, for instance—there is no permanent file corresponding to an instance, while for others—such as the device server—there is effectively no distinction between an instance and a permanent file. In the latter case, changes in the instance are immediately reflected in the underlying file.)

ing the processes associated with each file. Once an instance of a file has been created, it can be read, written, and released by means of library routines that implement the rest of the V-System's I/O protocol.

The authentication server. Each user of the V-System is issued an account name with a password and a user number. Security in the system is implemented by associating the user number with messages sent by processes created by the user. The mechanism discussed here[27] assumes that the sanctity of the distributed kernel cannot be violated. Messages sent by the kernel on one host cannot be tampered with by a user before they reach the kernel on another, nor can rogue versions of the kernel (or of the authentication server) be substituted for the authorized version. Although it may be unrealistic to expect such assumptions about security to be upheld in a commercial system, V is a research system running in a relatively friendly environment. We introduce a security mechanism into the system for the same reason we lock our cars: to keep the honest honest.

There are two principal resources in the V-System that are protected: files and hosts. The storage servers implement protection on files; the team server implements protection on hosts. To gain access to either, a user or an application must send a request for authentication to the authentication server. Only one authentication server is needed in the system, though it can be replicated. The request includes the account name and password of the user; the server checks them against an encrypted file on the storage server. If the passwords match, the server returns an indication of success to the process that sent the request, and *the kernel on that process's workstation associates the user number with the process* (Figure 19). From this point on, every message the newly authenticated process sends will include the user number, as will every message sent by any process it creates.

Users authenticate themselves by issuing a **login** command to an exec, giving a single password that will serve for all the servers in the system. When the password is verified by the authentication server, the server gives the exec the user's user number. When the exec asks for a new team to be created, the team server asks the kernel for the exec's user number. The team server, like other servers on the first team, runs with special privileges that allow it to set the user number of any process on the workstation. Using this capability, it sets the user number of the new team to that of the exec that sent the request.

A process can ask its kernel for the user number of any process that sends it a message. The storage server uses this facility to determine whether a request to open a file has come from a user having the appropriate authorization. It need only compare the user number with those on its access list for the file. Note that though any process can determine the user number of any other process, this capability does not give it any new privileges: The kernel is the only part of the system that can set the user number field in a message.

The team server can use the user number of its potential clients to determine whether it must start new programs on its host. A user can order his team server to refuse requests from any process with a user number other than his own. When the team server receives a message sent to the team server group that an application is looking for hosts, its reply indicates its host's lack of availability.

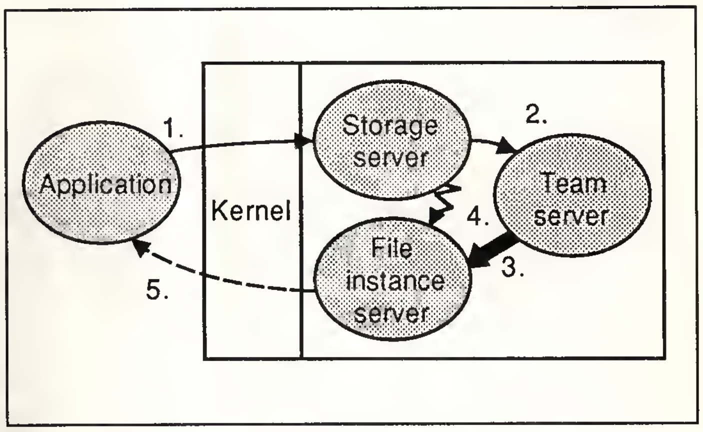


Figure 18. The storage server.

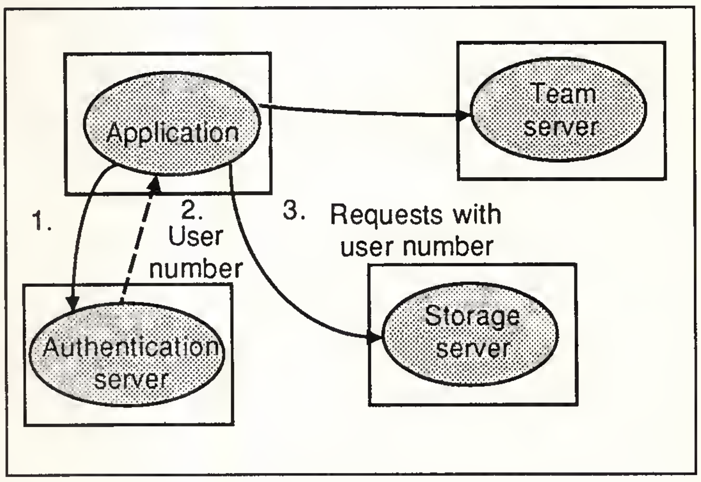


Figure 19. Authentication.

A whole distributed system—remote execution facilities

The first step toward creating a true system from a collection of hosts is allowing users to execute programs on hosts that are not being used.[23,24] The V-System does this by using the group communication facilities of the kernel, the team loading capability of the team servers, and a command parsing feature of the exec. If a user appends "@any" to an exec command, the exec foregoes the usual procedure of asking the team server on its workstation to load the program given in the command. Instead it sends a message to a process group made up of all the team servers running on

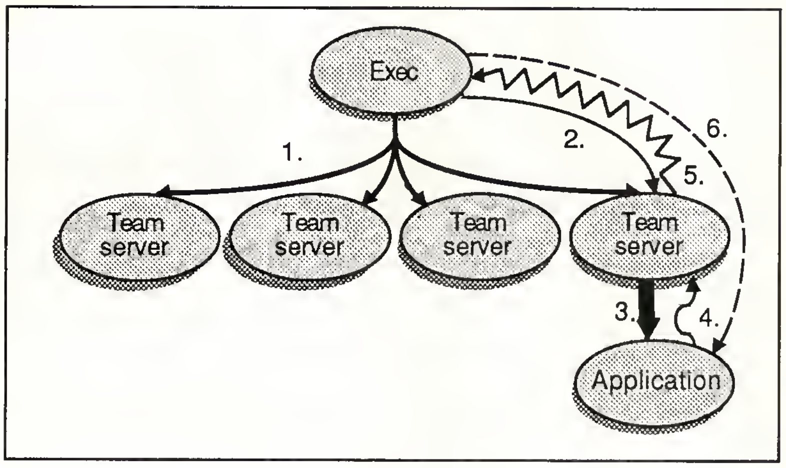


Figure 20. Starting a program remotely.

the workstations in the system (Figure 20). This message asks each team server to reply with its pid and an indication of the availability of its workstation for remote execution (Step 1 in the figure). The response includes the amount of memory available, the percentage of processor time used recently, and whether anyone using the machine has forbidden remote access.

The exec collects the responses until it decides it has found a suitable remote host. Then it issues the request (Step 2) it would normally have made to its own team server to the team server on the remote machine. From that point execution commences normally (Steps 3, 4, and 5). The distributed kernel allows the exec and new program to interact with each other normally. The team root message sent to the new team (Step 6) differs only in providing access to virtual terminals on the user's workstation instead of to ones on the host running the program. The I/O protocol, which uses the transparent message passing of the distributed kernel, makes it as easy for the application to write back to the user's screen as it does for it to write back to the local screen.

Using the same approach, we can take the next step: constructing programs which actually employ several hosts to work on a single task.[25] An application wanting to use more than one host can send a message to the team server group asking for volunteers (Step 1 in Figure 21). Upon receiving the number of hosts it needs (Step 2), the application can use the same routine the exec does to load the new team, put the parameters on its stack, and start it running (Step 3). If these "slave processes" are structured as servers dedicated to the "master" that started them, then the master can send them requests to perform various subtasks, receive the results in the slave's reply, and assign new tasks by sending new requests.

A heterogeneous system: the Unix servers

One of the services that would be too expensive to supply to each user of a distributed system is a complete mainframe and its operating system. To be able to use the complete capabilities of another system from a workstation running the V-System, one must build a program that runs on that system but simulates a V-System kernel and storage server. The Unix server is such a program. It provides access to Unix system services via the V-System kernel's interprocess communication primitives. To workstations running the V-System kernel, the Unix server appears to be a standard V-System storage server, one that provides Unix file access via the V-System's I/O protocol.

The Unix server maps the user number (Step 1 in Figure 22), which it trusts because it comes from the V-System

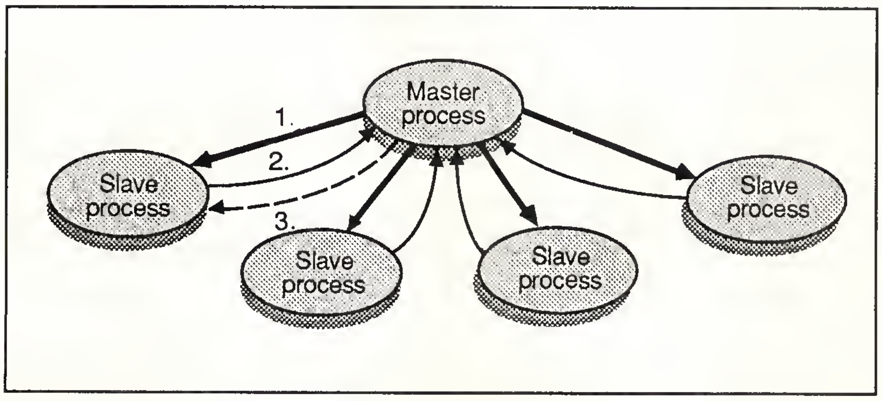


Figure 21. Master/slave parallel processing.

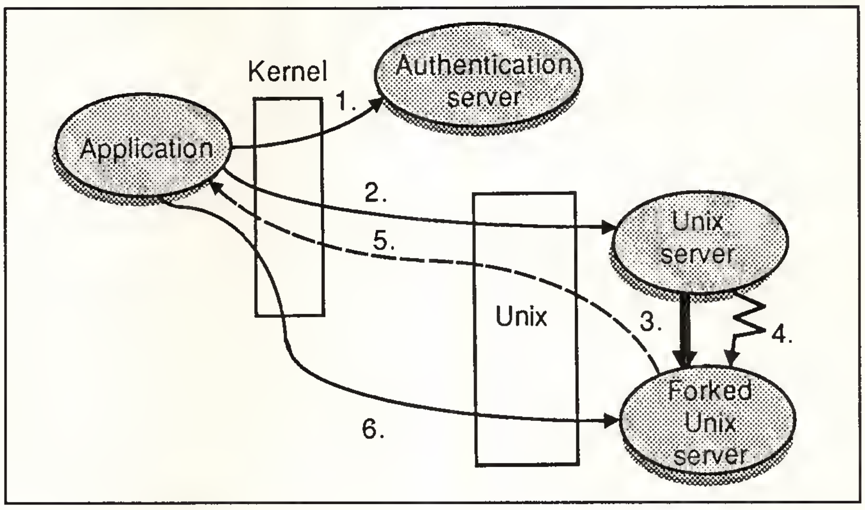


Figure 22. The Unix server.

kernel, to a Unix account accessible to that user (Step 2). It forks a copy of itself with the privileges of the user (Steps 3 and 4) and effectively assigns the copy a pid so that upcoming requests can be directed to it. The new pid is returned to the user/process (Step 5), which uses it for as long as it wishes to do file access (Step 6). If the forked Unix server does not receive requests for a time-out period, it closes the files that have been left open and destroys itself.

File access is not the most important service that a Unix machine can provide, however. Some applications run only on one operating system because they depend heavily on that system's facilities, or simply because they have not yet been ported to the V-System. The V-System's I/O protocol includes an option in CREATE_INSTANCE requests for specifying that the file be executed on its storage server when it is opened. The arguments of the program are passed in a segment with the request. In this way, the V-System can, in theory, run on any operating system as a guest, providing services otherwise not available to V-System users.

We believe that the V-System architecture serves as a good base for the development of distributed applications and for the further exploration of distributed systems issues. The kernel primitives have proven so far to be a firm foundation for the development of other system services. The group communication facilities in particular have surprised us with their simplicity and power in a number of applications. The client/server model, often used in conjunction with the I/O protocol, has allowed us to easily add a number of different services and devices. Finally, the modularity of the various system servers has enabled us to provide powerful capabilities to the user and yet allow for the frequent additions and changes that characterize a research environment. Though we make no claim that the V-System architecture is the only way to build a high-performance distributed system,[29] we do want to put it forward in support of our belief that such a thing can be done.

Acknowledgments

The design of the V-System is principally that of the Distributed Systems Group under the direction of David R. Cheriton and Keith A. Lantz. Other architects of the system are numerous—significant individual contributions are documented in the references. I extend special thanks to Joe Pallas for his suggestions and corrections and friendship.

Work on the V-System was supported by the Defense Advanced Research Projects Agency under contracts MDA903-80-C-0102 and N00039-83-K-0431, by Digital Equipment Corporation, by the National Science Foundation under grant DCR 83-52048, by the National Aeronautics and Space Administration under contract NAGW-419, by Bell-Northern Research, by AT&T Information Systems, by Philips Research, and by NCR, and by graduate fellowships from IBM, the National Science Foundation, Shell, and TRW. The Shell Companies Foundation provided support for the writing of this article.

References

1. K. A. Lantz, K. D. Gradischnig, J. A. Feldman, and R. F. Rashid, "Rochester's Intelligent Gateway," *Computer*, Vol. 15, No. 10, Oct. 1982, pp. 54-68.
2. D. R. Cheriton, M. A. Malcolm, L. S. Melen, and G. R. Sager, "Thoth, A Portable Real-Time Operating System," *Comm. ACM*, Vol. 22, No. 2, Feb. 1979, pp. 105-115.
3. G. Popek and B. Walker, eds., *The LOCUS Distributed Systems Architecture*, MIT Press, Cambridge, Mass., 1985.
4. R. Rashid and G. Robertson, "Accent: A Communication Oriented Network Operating System Kernel," *Proc. 8th Symp. on Operating Systems Principles*, ACM, Dec. 1981, pp. 64-75.

5. E. Lazowska et al., "The Architecture of the Eden System," *Proc. 8th Symp. on Operating Systems Principles*, ACM, Dec. 1981, pp. 148-159.

6. R. M. Needham and A. J. Herbert, *The Cambridge Distributed Computing System*, Addison-Wesley, Reading, Mass., 1982.

7. A. K. Jones, "The Object Model: A Conceptual Tool for Constructing Software," in *Operating Systems: An Advanced Course*, Springer-Verlag, New York, 1978.

8. A. Bechtolsheim, F. Baskett, and V. Pratt, *The Sun Workstation Architecture*, tech. report, Computer Science Dept., Stanford University, Stanford, Calif., Jan. 1982.

9. R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Comm. ACM*, Vol. 19, No. 7, July 1976, pp. 395-404.

10. *The Ethernet: A Local Area Network—Data Link Layer and Physical Layer Specifications, Version 1.0*, Digital Equipment Corp., Intel Corp., and Xerox Corp.

11. D. R. Cheriton and W. Zwaenepoel, "The Distributed V Kernel and Its Performance for Diskless Workstations," *Proc. 9th Symp. on Operating Systems Principles*, ACM, 1983.

12. W. Zwaenepoel, "Message Passing on a Local Network," PhD thesis, Stanford University, Stanford, Calif., 1985.

13. D. R. Cheriton, "The V Kernel: A Software Base for Distributed Systems," *IEEE Software*, Vol. 1, No. 2, Apr. 1984.

14. D. R. Cheriton and W. Zwaenepoel, "One-to-Many Interprocess Communication in the V-System," *Proc. ACM Sigcomm 84 Symp.*, 1984.

15. D. R. Cheriton and W. Zwaenepoel, "Distributed Process Groups in the V Kernel," *ACM Trans. Computer Systems*, Vol. 3, No. 3, May 1985.

16. D. R. Cheriton, "A Uniform I/O Interface for Distributed Systems," to appear in *ACM Trans. Computer Systems*.

17. E. J. Berglund and D. R. Cheriton, "Amaze: A Distributed Multi-player Game Program Using the Distributed V Kernel," *Proc. Fourth Int'l Conf. on Distributed Computing Systems*, 1984.

18. D. R. Cheriton, *The Thoth System: Multi-process Structuring and Portability*, American Elsevier, New York, 1982.

19. K. A. Lantz, D. R. Cheriton, and W. I. Nowicki, *Third Generation Graphics for Distributed Systems*, tech. report STAN-CS-82-958, Dept. of Computer Science, Stanford University, Stanford, Calif., Feb. 1983.

20. W. I. Nowicki, "Partitioning of Function in a Distributed Graphics System," PhD thesis, Stanford University, Stanford, Calif., 1985.

21. D. R. Cheriton and T. P. Mann, "A Decentralized Naming Facility," submitted to *ACM Trans. Computer Systems*.

22. K. A. Lantz, "An Architecture for Configurable User Interfaces," presentation at Working Conf. on the Future of Command Languages: Foundations for Human-Computer Interaction, IFIP, Sept. 1985. (Proc. to be published by North-Holland Pub. Co., Amsterdam, The Netherlands.)

23. M. M. Theimer, K. A. Lantz, and D. R. Cheriton, "Preemptable Remote Execution Facilities for the V-System," *Proc. 10th Symp. on Operating Systems Principles*, ACM, Dec. 1985.

24. M. M. Theimer, "Preemptable Remote Execution Facilities for a Distributed Computer System," PhD thesis, Stanford University, Stanford, Calif., 1986.

25. D. R. Cheriton and M. Stumm, "The Master-Slave Structure for Parallel Execution of Applications on a Workstation Cluster," in preparation.

26. D. R. Cheriton and P. J. Roy, "A Preliminary Report on the Performance of the V Storage Server," *Proc. ACM Computer Science Conf.*, 1985.

27. T. P. Mann and J. Juliao, "Authentication in the V-System," unpublished tech. report, Distributed Systems Group, Stanford University, Stanford, Calif.

28. K. A. Lantz and W. I. Nowicki, "Structured Graphics for Distributed Systems," *ACM Trans. Graphics,* Vol. 3, No. 1, Jan. 1984, pp. 23-51.

29. K. A. Lantz, W. I. Nowicki, and M. M. Theimer, "An Empirical Study of Distributed Application Performance," *IEEE Trans. Software Eng.*, Vol. SE-11, No. 10, Oct. 1985, pp. 1162-1174.

Eric J. Berglund is a PhD candidate in computer science at Stanford University. He is currently working with the Distributed Systems Group at Stanford under the supervision of David Cheriton and Keith Lantz.

Berglund received his BA in theater in 1978 and his MS in computer science in 1980, both from Michigan State University. He has served as a lecturer at Wayne State University in Detroit and is a member of the IEEE and the ACM.

Questions about this article can be directed to Berglund at the Computer Science Dept., Stanford University, Stanford, CA 94305.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 153 Medium 154 Low 155

A Benchmark Comparison of 32-bit Microprocessors

Thayne C. Cooper, Wayne D. Bell, Frank C. Lin, and Norm J. Rasmussen
Sperry Corporation

The benchmarking of four 32-bit microprocessors—the MC68020, the NS32032, the AT&T 32100, and the Intel 80386—revealed some specific reasons for the differences in performance that were observed.

The Research and Advanced Technology Group of Sperry Corporation has been responsible for evaluating microprocessors for the company since the days of eight-bit devices. Here, we present the results of our group's evaluation of four 32-bit microprocessors—the Motorola 68020,[1] the National 32032,[2] the AT&T 32100,[3] and the Intel 80386.[4] We used the *EDN* 16-bit benchmarks[5,6] to perform the evaluations, after modifying them for 32-bit operations and the test environment. The results of the evaluations are timings for the benchmarks.

Our study was an evaluation of the processors themselves, not of systems in which they may be used. Hence, we did not attempt to exercise all the features of the processors but instead restricted the evaluation to a fixed hardware environment and a fixed set of programs, to ensure that the processors would be compared under the same conditions.

We also benchmarked two 16-bit processors—the Motorola 68000 and the Intel 80286—to provide a comparison with the previous generation of devices. In the future, we will evaluate 32-bit processors that were not available at the time of this writing.

Benchmark environment

In preparing to carry out the evaluation, we decided to design a test system in which everything but the processor itself and its related parts would remain the same throughout the tests. We employed a bus that would support 32-bit addresses and data without affecting performance, devised a method for monitoring performance, and provided a memory card into which programs could be loaded and which the programs could use. The processors (along with their support logic) were placed on separate cards so that the testing of a processor would involve only the removal of one card and the replacement of it with another.

The test environment (right side of Figure 1) consists of a card cage (with a modified VMEbus backplane), a memory card (1M-byte dynamic or 32K-byte static RAM), and a processor card. A test tool card interfaces the test tool (left side of Figure 1) to the test environment's VMEbus and provides monitoring and other functions.

Test tool hardware. The test tool hardware comprises two parts: a Sperry PC/IT, an 80286-based computer system with standard peripherals, a color display, and a keyboard, and the PC/IT bus interface, a bus buffer card residing on the PC/IT bus and connected by a 64-pin cable to the test tool card.

The test tool card contains bidirectional, latched data and address registers, two breakpoint registers, comparators, a command register, a status register, and handshaking logic. Under command of the PC/IT, it can become a bus master having access to the entire VMEbus. When proper 32-bit data have been loaded, the test tool card releases the bus and allows the processor under test to run a program.

When a breakpoint is reached, the system stops, which in turn stops the timing being carried out by the PC/IT. In this way, actual processor operation times are measured. No test tool overhead is included in the measurements.

Test tool software. We used the PC/IT system to generate loadable code for each of the processors under test. To do this, we wrote, for most of the processors, cross-assemblers that run under the PC/IT DOS. We wrote these assemblers in C; they produce nonrelocatable ASCII loadable files. We used assemblers instead of compilers so we could produce efficient code for each of the processors.

We wrote another PC/IT program to provide the interface to the test environment. Under this program's control and through commands from the PC/IT keyboard, the benchmark code is loaded from the ASCII files into the memory board in the test environment, changes to or inspection of the memory are made, breakpoints are set for execution, and the test environment is reset and started. This program also performs functions to help control and debug test programs.

By using the timer in the PC/IT-based test tool hardware and the breakpoint capability provided by the test tool card, we obtained the timings given in this report. During a timing, the test tool is monitoring the bus for the address set in its breakpoint registers but is not making requests for the bus.

Our study was an evaluation of the processors themselves, not of systems in which they may be used.

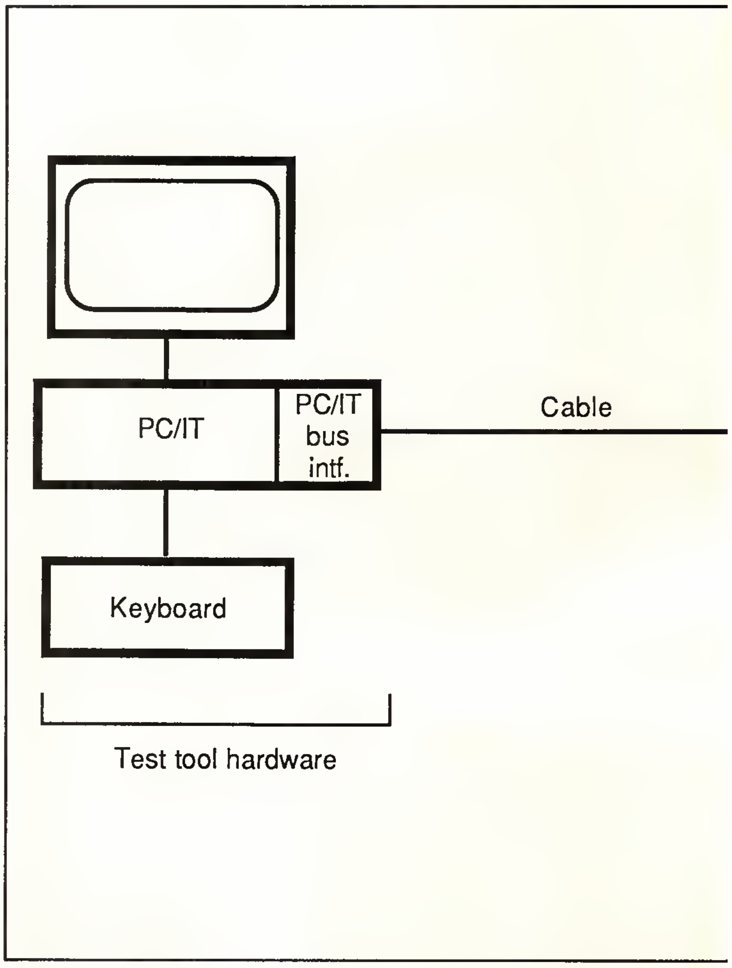
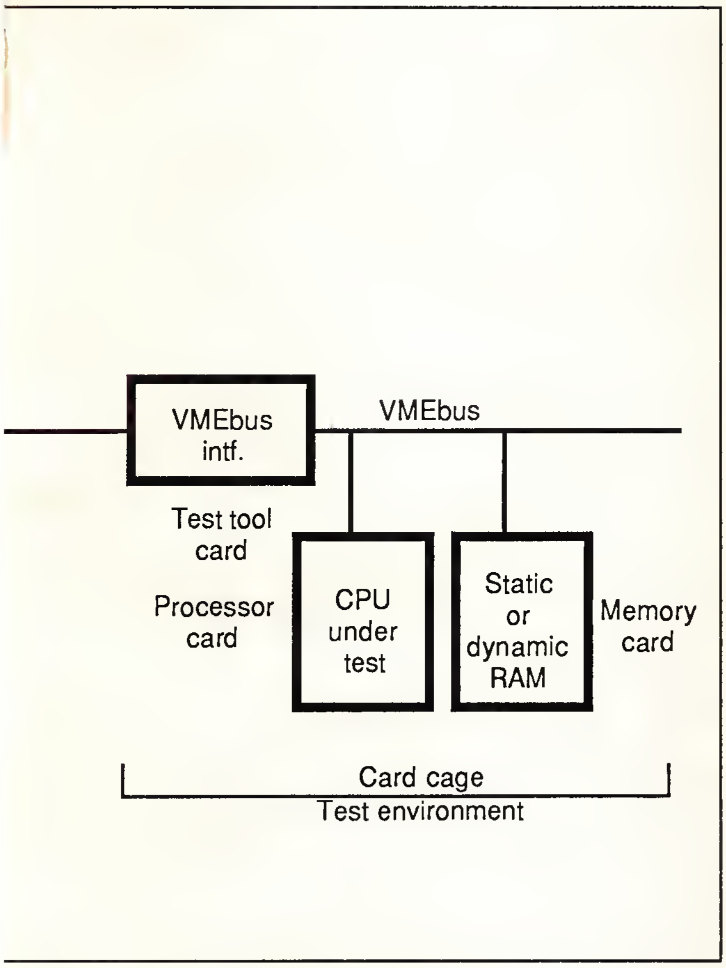


Figure 1. Block diagram of the test system. The test tool, a Sperry PC/IT, is connected to the test environment, a card cage utilizing a VMEbus

VMEbus. The VMEbus was defined by three companies—Motorola, Mostek, and Signetics.[7] It utilizes two 96-pin Eurocard connectors, which allow a full, 32-bit, nonmultiplexed data bus and up to 32 bits of address. It also accommodates control signals and I/O.

We found one deficiency in the VMEbus. The 32-bit processors we wanted to evaluate allow the writing of data on all byte boundaries and lengths, but the VME specification does not accommodate all the combinations the processors can produce. Therefore, we modified the VMEbus by taking four of the six VMEbus address modifier bits and defining them as four byte enables: BE0, BE1, BE2, and BE3 replace AM0, AM1, AM2, and AM3. With this change, the VMEbus allows odd byte boundaries and odd



backplane and containing a processor card and a memory card, through a cable with interface cards on each end.

numbers of bytes to be written easily. The 32-bit processors we chose for evaluation require only minor adjustments to work with a VMEbus thus modified.

Except for making these modifications, we used the bus as specified—that is, we used the specified timing, termination, signals, handshaking, power, connectors, and cards.

Memory cards. We built two memory cards, one using 256K × 1 dynamic RAMs and the other using static devices.

On the dynamic RAM board, we used parts having a 150-nanosecond access time and a 300-nanosecond cycle time and used the National Semiconductor 8409 DRAM controller. With this board, a typical machine cycle is 600 nanoseconds long, or 775 nanoseconds long if refresh is waited for.

On the static RAM board, we used 45-nanosecond devices. With buffering and propagation delays, the board becomes a 70-nanosecond (from processor request to data at the processor) memory. Our intent was to provide a no-wait-state memory that would appear to the processor as an external cache with no misses. As it turned out, because of internal processor and support chip delays, bus delays, and READY/WAIT/DTACK handling, we needed to add a one-clock-cycle wait to run the system with the static RAM board. As a result, we obtained the following machine cycle times for a memory read:

- 16-MHz 68020—230 nanoseconds,
- 18-MHz 32100—220 nanoseconds,
- 10-MHz 32032—260 nanoseconds, and
- 16-MHz 80386—190 nanoseconds.

These times are consistent with the processors' data sheets.

Through careful design of the entire test system, we could have eliminated the delays and thus the one-clock-cycle wait. By eliminating this wait for the 80386, we could have reduced a read to 125 nanoseconds, resulting in a 9.7-percent improvement in speed over the benchmark mean we report later. We did not make this change, however, because of difficulties in making it work with the dynamic memory, but we mention it to illustrate the speed increase one can obtain.

We organized the dynamic RAM as 256K bits × 32 bits, with byte enables mechanized. We organized the static RAM as 8K bits × 32 bits, with 16K bytes at low locations to accommodate Motorola-type starting addresses and 16K bytes at high locations to accommodate Intel-type starting addresses.

Processor cards. A processor card contains a processor and logic for interfacing to the VMEbus. We added arbitration logic to the card if the processor did not provide that function. This logic consists primarily of the processor's existing bus request and bus grant lines adapted to the VMEbus. Since programs are loaded from the test tool into RAM, we did not put any ROM on the processor cards.

We added logic to the cards for the two 16-bit processors we evaluated that allows them to select the proper half of the bus to work on.

On the 68020 card, a memory cycle starts when the address strobe (AS) signal becomes active. Note that we did not use early address strobe (ECS or OCS) in our design, although its use can increase the speed of the processor.

We designed a test system in which everything but the processor and related parts would remain the same throughout.

The VMEbus is optimized for 16-bit, 68000-type processors. As a result, we had to doctor some of the signals from the other types of processors and support chips to make sure they met VME specifications. Among these were READY (so it would look like DTACK), the address strobe, and the data strobes. (However, the 32032 and the 80386 provided byte enables that mapped directly to our modified VMEbus.) The changes we made to the signals were not major ones; we estimate that they slowed performance by less than five percent.

No "388" chip for bus control is going to be manufactured to support the 386. Intel feels this kind of chip is a speed limiter and that the user can optimize his system with his own design. Intel feels the same way about the handling of READY—the 82384 clock generator chip does not have a READY signal input. Thus, we had to synchronize READY to CLK2.

The 80386 has an ADS# (address strobe not) signal output that is not synchronized to the clock. The 82384 chip performs this synchronization but adds a significant 50-nanosecond delay. This was the primary reason we had to add a one-clock-cycle wait for fast static memory access. ADS# may be used directly if clock synchronization is not required.

The 32100's design is a straightforward one. The only difficulty we encountered—a slight one—was with the 32100's ABORT line. When the 32100 is using its cache and does a jump or branch, it starts an instruction fetch from RAM. But if it finds the instruction in its cache, it aborts the memory cycle. Our memory card design does not implement an aborted memory cycle, however. We solved the problem by adding logic to the 32100 processor board to provide a delay of up to three additional cycles after an aborted cycle.

It is obvious that a cache helps each processor that has one and would help those that do not. Having a cache on chip, even if it is a small cache, does contribute significantly to processor speed.

The benchmarks

The test programs we used were originally written for the 68000, the 8086, and the Z8000.[5,6] We modified some of the 68000 and 8086 programs to take advantage of the new instruction and address modes of the 68020 and 80386. We wrote new versions of the programs for the 32032 and the 32100. We had the same person modify or write all the tests so we could be sure that the same algorithms would be used for all the processors.

Each of the tests written for the 68020 had a few extra instructions at the beginning to test a fixed byte for nonzero. If the byte was set to nonzero, the cache was enabled. The extra instructions did not add significantly to the overall times of the tests, since the times were taken in seconds. In the 32100 programs, the initial control word was changed to enable the cache. Thus, we could run the tests with and without the cache simply by changing one byte from the test tool before starting the processor.

The tests are subroutines that were called a number of times before they fell through to a breakpoint instruction. This was done so the times obtained would be large and not affected significantly by any inaccuracies in the timing done by the test tool code. In our evaluation, we set the number of times the subroutines would be called to a figure that would yield values close to 10 seconds. These values served as the basis of our comparisons.

We used only five of the *EDN* programs: E, F, H, I, and K. We added loops to the code of each to give us longer times to measure; the time added by these loops is included in the total time for each test. In some cases, we saved more registers in our version than were saved in the original code, and this made the looping easier.

The five tests represent a variety of tasks:

- Test E is a character-string search routine. It looks for a 15-character-long string inside a 120-character-long string. This routine looped 24,577 times.
- Test F is a bit test, set, and reset routine. It works on a string of 128 bits and goes through a sequence to test, set, and reset three of them. This routine looped 65,536 times.
- Test H is a linked-list insertion routine. It inserts five records into an empty list. This routine looped 24,577 times.
- Test I is a quicksort routine. It takes 100 records and sorts them. It moves the records to be sorted to a new area of memory and sorts them out of that area. It repeats this process 257 times.
- Test K is a bit-matrix transposition routine. It transposes a 7-bit × 7-bit matrix. This routine looped 20,481 times.

Benchmark results

The results of our benchmarking are given in Tables 1 and 2. Table 1 shows the time taken by each processor to run each benchmark with the dynamic RAM board in the system. Table 2 shows the time taken by each processor to run each benchmark with the static RAM board in the system. Table 3 summarizes the results shown in Tables 1 and 2. It gives a mean of the results of the five tests for each processor and provides a processor comparison ratio, with the fastest processor, the 68020 with cache enabled, set to 1.00.

It is obvious that a cache helps each processor that has one and would help those that do not. Having a cache on chip, even if it is a small cache, does contribute significantly to processor speed. Even the 68020 and 32100, which have a cache, would be helped by an external cache if they were in a system running large programs.

In some of the tests, some of the processors were able to work from their registers, while others with fewer registers were forced to keep some values on the stack. This means they had to take time to read and write information instead of using it.

Some addressing modes helped the operations—namely, the auto increment and decrement modes and the register indirect without displacement mode.

The 80386's repeated string instructions enabled that processor to run the tests that could use them very fast.

The 32032's instructions set very few condition flags. One flag that most of them do not set, but that is very useful, is the zero condition. The lack of this flag reduces the 32032's efficiency. To determine if an operation such as subtraction ends with a zero, for example, one must compare, before testing, for the zero condition. This takes an extra instruction and time.

The 32100 made superfluous instruction fetches at times. In one loop, for example, three out of nine long words fetched were not needed. This was probably an extreme case.

Alignment of code and data on long-word boundaries enhances the performance of 32-bit processors significantly. (It is a less important consideration with 8- and 16-bit processors.) We obtained the times reported here after we aligned data on long-word boundaries.

Our test setup was not the best for measuring absolute performance, if absolute performance means the performance that can be obtained by using every conceivable design trick in the test system. Some memory could have resided on the processor card so more concurrency could have been achieved, for example, and the static RAM could have been designed to provide no-wait-state operation. However, our aim was to create a test environment in which we could measure relative performance—that is, compare the performances of processors that had executed the same set of tasks. In this our test setup was successful. ■

Table 1.
Benchmark times (dynamic memory). These are absolute times, in seconds, for each processor executing each benchmark. (N = no cache, C = cache.)

		Benchmark				
	MHz	E	F	H	I	K
80286	(10)	4.89	13.63	6.59	11.20	19.39
80386	(16)	3.57	5.16	3.63	6.86	6.20
68000	(8)	13.73	14.61	8.79	12.08	16.59
68020 N	(16)	8.02	5.55	3.84	5.65	4.78
68020 C	(16)	3.84	2.47	2.14	2.75	3.02
32032	(10)	12.52	13.07	6.21	8.57	13.07
32100 N	(18)	16.81	8.84	5.05	8.57	9.17
32100 C	(18)	6.75	4.29	2.74	3.63	4.45

Table 2.
Benchmark times (static memory). These are absolute times, in seconds, for each processor executing each program. (N = no cache, C = cache.)

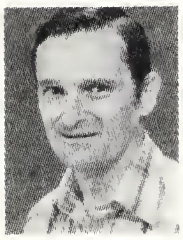
		Benchmark				
	MHz	E	F	H	I	K
80286	(10)	3.18	6.81	3.46	5.98	10.27
80386	(16)	1.92	2.53	1.53	3.18	3.68
68000	(8)	8.79	9.67	5.60	7.69	11.37
68020 N	(16)	3.74	2.74	1.87	2.69	2.64
68020 C	(16)	2.52	1.98	1.26	1.92	2.25
32032	(10)	11.04	10.05	4.83	6.59	11.65
32100 N	(18)	9.28	4.72	2.52	4.45	4.84
32100 C	(18)	6.04	3.07	1.81	2.97	3.46

Table 3.
Mean of the test times for each processor with each memory, and the processor comparison ratio derived from the means.(Times are in seconds; N = no cache, C = cache.)

		Dynamic memory		Static memory	
	MHz	Mean	Ratio	Mean	Ratio
80286	(10)	11.14	3.92	5.94	2.98
80386	(16)	5.08	1.79	2.57	1.29
68000	(8)	13.16	4.63	8.62	4.33
68020 N	(16)	5.57	1.96	2.74	1.38
68020 C	(16)	2.84	1.00	1.99	1.00
32032	(10)	10.69	3.76	8.83	4.44
32100 N	(18)	9.69	3.41	5.16	2.59
32100 C	(18)	4.37	1.54	3.47	1.74

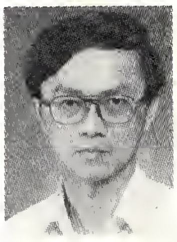
References

1. *MC68020 32-bit Microprocessor User's Manual*, 2nd ed., Prentice-Hall, Englewood Cliffs, N.J., 1985.
2. *NS32032-10 High Performance Microprocessor*, National Semiconductor Corp., Santa Clara, Calif., Feb. 1984.
3. *WE32100 Microprocessor Information Manual*, AT&T Technologies Inc., Morristown, N.J., Jan. 1985.
4. *80386 High Performance Microprocessor with Integrated Memory Management*, Intel Corp., Santa Clara, Calif., Oct. 1985.
5. R. D. Grappel and J. E. Hemenway, "A Tale of Four μPs: Benchmarks Quantify Performance," *EDN*, April 1, 1981, pp. 179-265.
6. W. Patstone, "16-bit-μP Benchmarks—An Update with Explanations," *EDN*, Sept. 16, 1981, pp. 169-203.
7. *VMEbus Specification Manual*, Rev. C, VMEbus Manufacturers Group, Feb. 1985. (The VMEbus has also been standardized as the IEC 821 bus and the IEEE 1014 bus.)



Thayne C. Cooper is a staff programmer with the Research and Advanced Technology Group of Sperry Corporation in Salt Lake City, Utah. He joined Sperry in 1969 after receiving a BS degree in math from Brigham Young University.

Wayne D. Bell is a staff engineer with the Research and Advanced Technology Group of Sperry Corporation. He joined Sperry in 1960 after receiving a BSEE degree from Utah State University.



Frank C. H. Lin is a staff engineer with ESL, Inc., in Sunnyvale, California. He worked for CalComp Electronics, Inc., from 1975 to 1977 and for Sperry Corporation from 1978 to 1986. He was manager of Sperry's Research and Advanced Technology Group when the work reported here was performed. Lin received a BSEE in 1973 from the National Taiwan University, an MSEE in 1978 from Utah State University, and a PhD in computer science in 1985 from the University of Utah.

Norm J. Rasmussen is a design engineer with the Research and Advanced Technology Group of Sperry Corporation. He joined Sperry in 1982 after receiving a BSEE from Utah State University.

Questions about this article may be directed to Thayne C. Cooper at Sperry Corporation, Computer Systems, 322 North 2200 West, Salt Lake City, UT 84116.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 150 Medium 151 Low 152

MicroLaw

by Richard H. Stern/Law Offices of Richard H. Stern/2101 L Street NW, Suite 800/Washington, DC 20037

Reverse engineering of chips

Part II: A case example

In the first part of this series I discussed the events and decisions leading to the 1985 passing of the reverse-engineering provisions of the new Semiconductor Chip Protection Act. Here is a case history that may suggest either chip piracy or legitimate reverse engineering. I invite readers to submit their opinions of this example. I welcome any other examples readers care to send me; I will comment on these cases in the December issue.

The following is an actual example of a chip rights controversy. The identities of the parties and the chips are concealed. In this case, manufacturer A designed and marketed the chip (designated here as Alpha) as an automotive part. Figure 1 illustrates the chip in place, in an automotive fuel injection system. Later, manufacturer B decided to market its version of the chip, Beta. Alpha and Beta are compatible in form, fit, and function, and in fact they compete for the same business. When Beta began to divert sales away from Alpha, A complained to B of alleged chip piracy. But B responded that it had merely engaged in permissible reverse engineering, as evidenced by a substantial paper trail.

Figure 2 is a circuit schematic of Beta. (The schematic of Alpha is slightly different.) Figures 3 and 4 are die photographs of Alpha and Beta, respectively. Usually, it is possible to appreciate why a chip is laid out in a particular way, and to determine what aspects of the topography may be dictated by function, only by study of the schematic as well as the die.

Comparison of these die photos suggests considerable similarity in the respective "floor plans" of the two chips. It is clear, moreover (1) that Beta is not a photographic reproduction or "clone" of Alpha, and (2) that B laid out Beta's topography after studying Alpha. Probably most informed observers will agree that substantial arguments can be made here in support of each manufacturer's position. A comparison of the salient features of the two chip layouts follows.

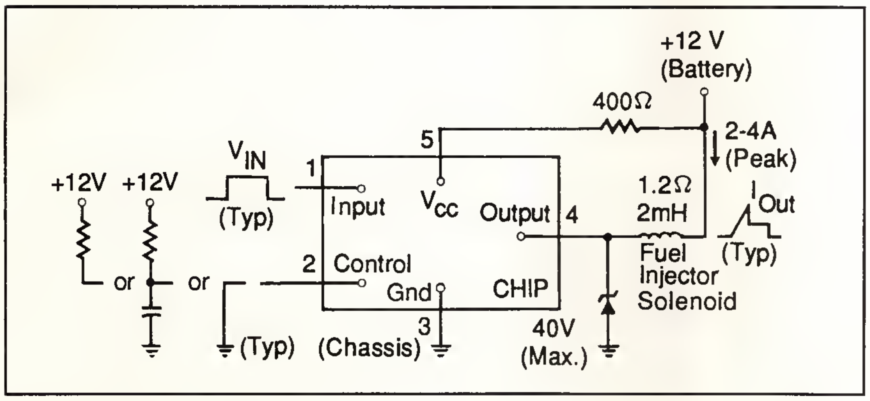


Figure 1. Alpha schematic.

Input/output pads

Most of the input/output pads are in the same locations in both chips. To some extent, the input/output pad locations are dictated by the need to be near their respective pins; the connections should be reasonably short and not cross over one another.

Thus, the upper-left input pad had to be adjacent to the giant Zener diode that it feeds, but the diode did not have to a be in the upper-left corner. It could have been almost anywhere along the upper edges of the chip. Similarly, the upper-right input pads could have been located in any convenient position along the upper edges of the chip. Thus, these similarities in "floor plan" were not functionally dictated and were unnecessary.

The array of four pads (in an upside-down T pattern) at the lower center of the die photographs connects to the OUTPUT and GROUND. The leftmost and rightmost pads are wired together and connect to GROUND, a tab which is screwed to the chassis of the car. The middle two pads are connected in parallel to the collectors of the power output stage (discussed in the following paragraph) on the chip and to the OUTPUT.

The ground output pads in the lower part of the chip had to be placed near the emitter resistor metallization paths (the stepped, skyscraper profile structures), so that their location in the floor plan depends on the location of the emitter resistor metallization paths.

The collector (OUTPUT) pads had to be placed on the giant collector's metallization, but the pads of Beta could have been placed differently from those of Alpha. For example, the pad at the center of the upside-down T pattern could have been lowered to a point near the bottom edge of the die, or the T pattern could have been turned rightside up.

Power transistors

There is a large, winding-path structure occupying the lower half of each die, which resembles two fat I's flanked on either side by an I broken in the middle. At the right side of each die a pair of rectangular pads is inserted between the parts of the broken I. At the left side of each die a winding-path structure is inserted into the broken I.

The large, winding-path structure may be considered a single, giant power transistor with one huge collector and over 100 base-emitter pairs (or it may be regarded as 100+ power transistors in parallel). The huge collector is the light gray (speckled) area. The emitters and bases interdigitate. The bases are fed by a driver transistor connected to the power transistor in a Darlington configuration. The base of the driver transistor is fed a signal from the upper part of the die. The structure may also be regarded as 100+ parallel, interconnected power transistors. In any case, this part of the die corresponds to the upper-right part of the chip schematic, where two transistors within a gray screen are shown as connected by dotted lines. The power transistor stage delivers current (2 to 4 amperes) to the solenoid for the fuel injector, in response to signals from the circuitry that occupies most of the upper part of the die (and most of the rest of the chip schematic).

The interior of each I or half-I consists of two main groups of parts. The larger group of parts, which resembles interdigitated fingers, is the set of base-emitter pairs in the power transistor stage. The smaller, boxlike parts that are located on the side of the I's adjacent to the stepped structures, are "ballast resistors," which tend to equalize current flow among the 100+ emitters. Each emitter is connected to a separate, approximately 0.1-ohm, ballast resistor, which is then connected to ground.

That B's layout of the Beta chip follows A's layout of the Alpha chip in placing the giant power transistor structure at one end of the die is not significant. Since the structure occupies more than half of the die, it had to be at one end or the other. If counsel turns the chip the right way, the power transistors will always be at the same ends. This is one of the most striking similarities in the floor plan, at least to the casual observer, but it is not of much real significance.

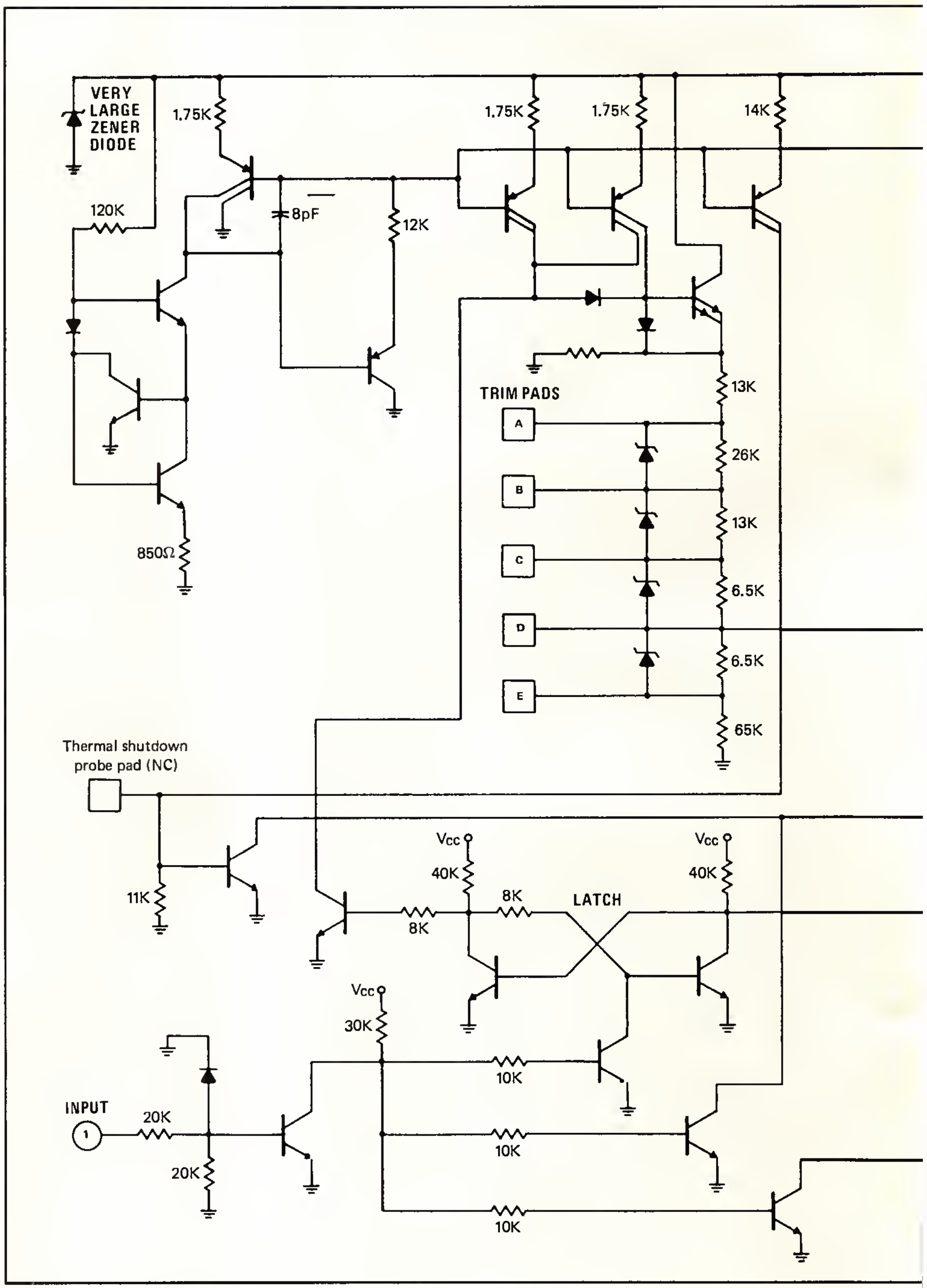
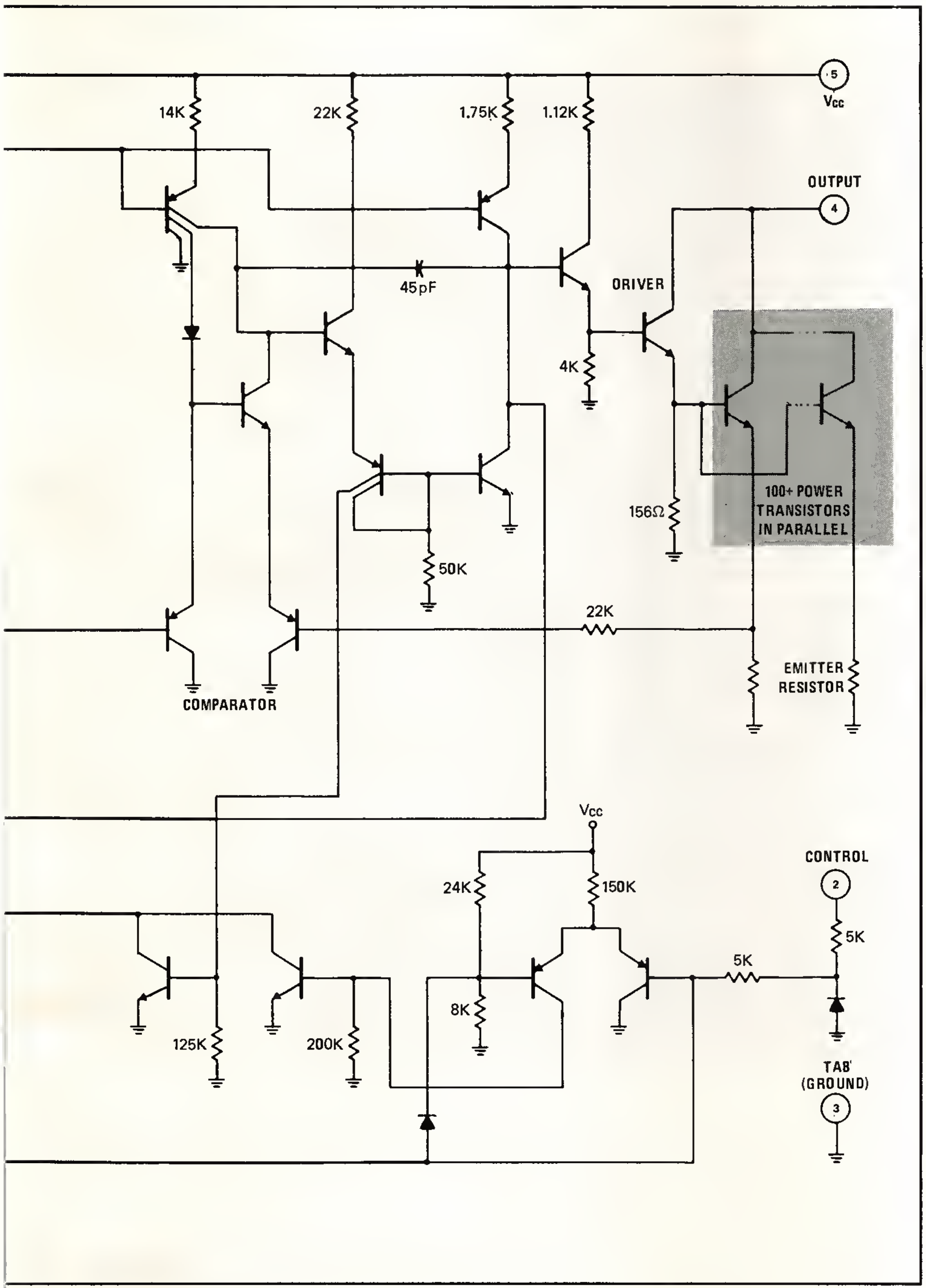


Figure 2. Beta schematic.

The back-and-forth winding paths (interdigitations) in the power transistors are characteristic of such circuitry and are conventional. Probably, the total lengths and widths of the paths, and their spacing, are dictated by engineering considerations and cannot be altered substantially. The general symmetry of the power-output stage reflects a purpose to keep the emitters and their ballast resistors at a more or less uniform temperature, so that approximately equal current is maintained in each of the 100+ emitters, sharing the loading equally and overloading no one emitter. Again, the similarity in appearance here is largely deceptive and does not show piracy.



If only this part of the chip were involved, there would be a very strong case for reverse engineering.

The approximately square structure inserted into the space between the upper and lower parts of the broken I at the lower-left edge of the die is an approximately 65K-ohm resistor, shown in the chip schematic under TRIM PAD E and to the left of the COMPARATOR. In Beta, this resistor may be trimmed to a value between approximately 40K and 65K. The approximately 65K resistor goes to one side of a comparator stage. The other side of the comparator stage senses the voltage across one of the 100+ emitter ballast resistors and thus measures the current output of the entire device. The 65K resistor is located amidst the emitters and their ballast resistors to subject it to the same temperature changes they meet, which has the functional effect that the shape of the current wave delivered at the output remains the same regardless of temperature changes.

The 65K resistor was inserted among one set of emitters and their ballast resistors to place it at the same temperature as the latter. This placement required a break in one bank of emitter-base pairs, and the bank on the opposite edge of the die was similarly broken to preserve thermal symmetry. The foregoing characteristics of the layout reflect functional and engineering considerations. Accordingly, the similarities do not cause infringement of mask work rights.

Nevertheless, the floor plan of the bottom part of the die could have been arranged in a number of ways. For example, B could have rotated the whole pattern 90 degrees to make it different, without creating routing problems or violating good engineering practice. Probably, other layout changes were feasible. That this did not happen possibly was due to the designer's concern whether the customer would accept the claim that Beta was compatible in form, fit, and function with Alpha.

The question that Beta's manufacturer B may therefore properly ask is whether the SCPA requires B to make such changes for the sake of change. Assuming that, as claimed, the layout of Beta is supported by a wheelbarrow full of "paper trail" and many dollars worth of timesheets, it would appear to be enough that the two dies are not substantially identical. The question then reduces to whether B's duplication of this aspect of the floor plan of Alpha, taken together with the rest of the similarities in design of the two chips, provides enough sameness to make the two dies substantially identical.

The 65K resistor had to be among the emitter resistors to carry out its temperature-compensation function. The internal layout of this part is dissimilar in the two dies. Indeed, Beta has certain technical improvements over Alpha in this part of the chip (an ability to trim the value of resistance, so as to "fine-tune" the comparator) so that if only this part of the chip were involved, there would be a very strong case for reverse engineering. Presumably, in any litigation B's counsel would place great stress on this aspect of the design.

The power-transistor part of the die probably reflects the major part of the layout design, in terms of time and expense. If it were possible for A, the manufacturer of Alpha, to register this cell of the chip alone, without the other parts where there is less similarity, manufacturer A would gain a considerable advantage in any mask work litigation. This is because the similarity between Beta and what A had registered would then appear much more substantial, since the dissimilarities in the other parts of the die would not detract from the similarities of the power circuitry. As indicated elsewhere (*IEEE Micro*, Feb. 1986, p. 72; id., June 1985, pp. 73-74), however, the Copyright Office's regulations in most instances do not permit registration of a cell when the cell has later been incorporated into a full IC.

Emitter metallization

The four stepped structures in the bottom half of the die, between the I's and the broken I's, which look like profiles of skyscrapers, are metallic current paths to ground for the 100+ emitters of the output power transistor and their 100+ respective emitter-to-ground resistors. Beta's structures are right-left mirror reflections of Alpha's and are slightly less wide.

The structures are stepped in the layout because design considerations call for equal voltage drops across the 100+ emitter-to-resistor-to-ground paths. Several amperes of current flow through this metallization, so that it is important that the width of the metal path be tapered to equalize voltage drops. As current increases, a wider path must be provided. The width of the steps is dictated at each point by the magnitude of the current at that point. Thus, the shape of the skyscraper profiles is functionally dictated and not significant for SCPA purposes, despite any similarity of appearance.

However, the location of the structures is dictated only by their having to be between the emitter resistors and ground. Thus, where the skyscrapers are located on the die depends on how the rest of the power transistor array is laid out. As indicated above, this is arbitrary.

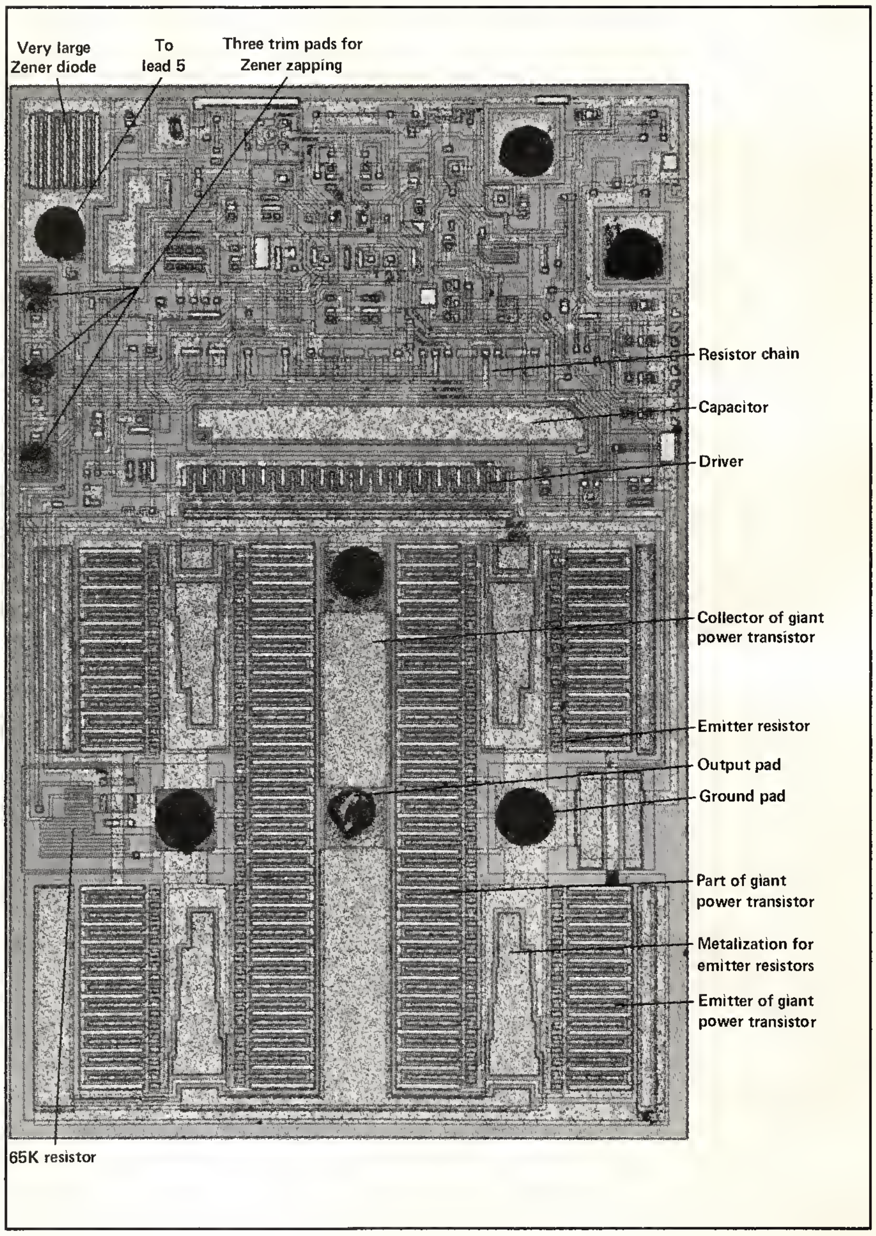

Very large Zener diode
To lead 5
Three trim pads for Zener zapping
Resistor chain
Capacitor
Driver
Collector of giant power transistor
Emitter resistor
Output pad
Ground pad
Part of giant power transistor
Metalization for emitter resistors
Emitter of giant power transistor
65K resistor

Figure 3. Die photograph of Alpha chip.

Driver

The narrow back-and-forth winding path just above the middle of each die and above the giant power transistor is a driver transistor that feeds the 100+ bases of the power output stage described above. The driver is shown in the upper-right portion of the chip schematic, directly to the left of the power output stage.

The driver has to be near the output power transistor array. Probably, there is no more convenient place to put it than directly above the array of power transistors. The internal structure of the driver appears to be conventional, and the Beta design actually differs slightly from Alpha's.

Accordingly, this aspect of the design does not contribute to infringing similarity.

Did B "needlessly"

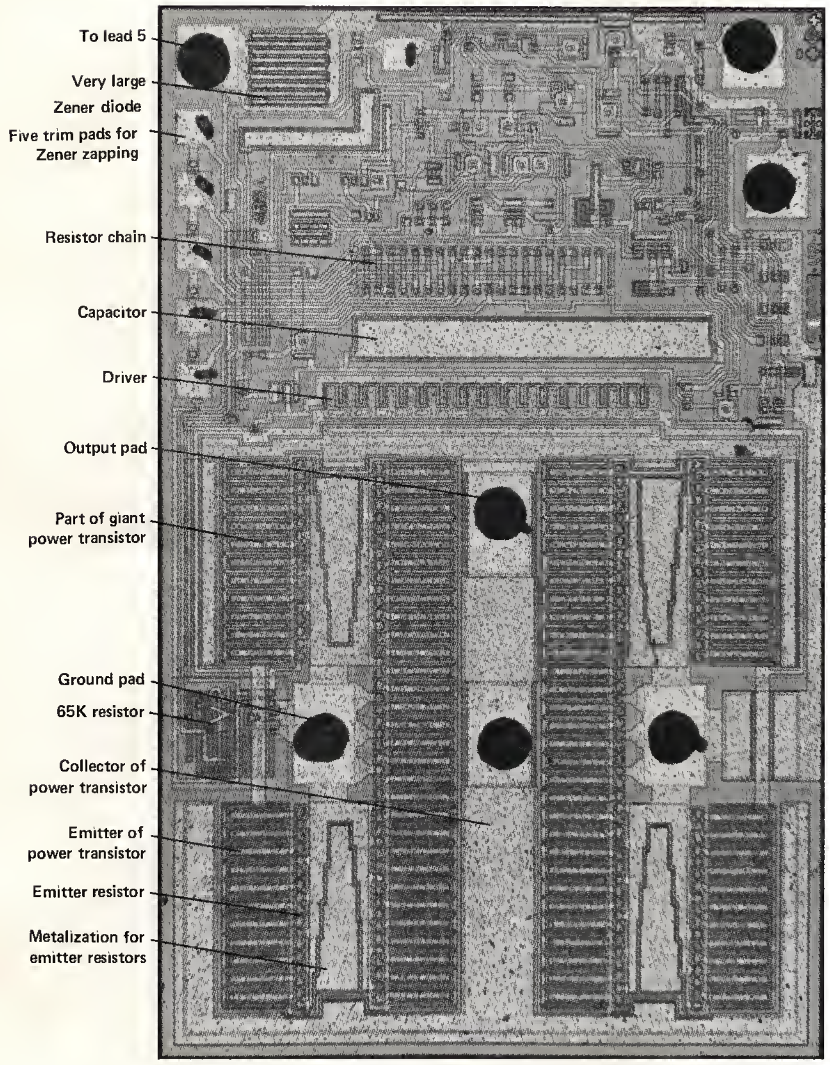

To lead 5
Very large Zener diode
Five trim pads for Zener zapping
Resistor chain
Capacitor
Driver
Output pad
Part of giant power transistor
Ground pad
65K resistor
Collector of power transistor
Emitter of power transistor
Emitter resistor
Metalization for emitter resistors

Figure 4. Die photograph of Beta chip.

copy the floor plan?

Capacitor

Immediately above the driver is an approximately rectangular padlike area. It is a metallic capacitor plate. The capacitor is shown in the upper center of the chip schematic, marked 45 pF.

The capacitor does not have to be near the driver, although it is right over it in both dies. The area, but not the shapes, of the metal plates of the two capacitors is dictated by technical considerations.

The capacitor and its internal structure appear to contribute somewhat to infringing similarity. But taken by itself, this is not a very substantial appropriation of A's layout.

Resistor chain

On the upper-left edge of the die, Alpha has three pads and Beta has five. The pads each connect a string of resistors (shaped like **U**'s in Alpha and like **I**'s in Beta). These resistors have Zener diodes in parallel with them (and are shown in the chip schematic near its center) to the right of the notation "TRIM PADS." The Zener diodes act somewhat as the inverse of a fuse, and they are selectively "zapped" (shorted out by a current overload) to trim the resistor string to a desired resistance value. Alpha employs a different "Zener zapping" technology from that of Beta, so that it requires fewer pads. That Beta has five pads requires it to be longer than Alpha, which increases chip area.

Probably, one should consider the location of this circuitry within the floor plan of the chip and the internal structure of the Zener-zapping sections differently.

Clearly, Beta's five-pad layout is not a copy of Alpha's three-pad layout. Indeed, the different Zener-zapping technologies led to different layouts within the pad module.

As to overall chip floor plan, however, a different conclusion may be drawn. The pads for the resistors and the Zener diodes probably need to be near an edge of the die, so that they are readily accessible for zapping. But which edge of the die is used is arbitrary. Thus, B "needlessly" copied the floor plan of Alpha in placing the pads at the upper-left corner of the die.

Similar considerations apply to the resistors. The internal structures are different, but the location within the floor plan of the total die is "needlessly" imitative. This aspect of the layout shows apparent copying of the floor plan, but the substantiality of the particular copying within the total layout of the chip, considering all elements, is not very great.

Zener diode filter

Near the upper-left corner of each die is a set of back-and-forth winding paths (approximately 10 in Beta, eight in Alpha). These paths form a giant Zener diode, used as an input filter and voltage reducer. In conjunction with an externally connected resistor, which is shown in Figure 1 as 400 ohms connected between lead 5 and +12 volts, the Zener diode reduces the 12 volts from the automobile's battery to approximately 7 volts and filters or regulates it to a substantially constant value. The giant Zener diode is shown in the chip schematic (Figure 2) at the upper-left

continued on page 75

MicroNews

MicroNews features information of interest to professionals in the microcomputer/microprocessor industry. Send information for inclusion in MicroNews one month before cover date to Managing Editor, IEEE Micro, *10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.*

Technology can be fun!

Personal Computer Pioneers Day held at The Computer Museum in Boston this June honored industry pioneers for their contributions to the PC revolution. Computer developers John Blankenbaker, Thi Truong, Ed Roberts, and Lee Felsenstein contributed models of their early machines to the museum's collection of historical artifacts.

John Blankenbaker is credited with inventing the earliest known personal computer, the Kenbak-1, in 1971 as a low-cost hands-on tool for the classroom. Thi Truong's French-built Micral, marketed in 1975, was acknowledged as one of the first successful PCs marketed outside the US.

MITS founder and president Ed Roberts discussed the highly successful Altair 8800, which he designed in 1975. Lee Felsenstein, an original member of the Homebrew Computer Club and designer of the Osborne 1, remarked that creative energy will continue to be driven by "the opportunity to get your hands dirty with technology and still have fun."

The Computer Museum is devoted solely to computers and computing; it is an independent nonprofit institution with an international audience and membership.

Update: operating system standard

The IEEE Computer Society 1003 Working Group's recently released *IEEE 1003.1 Trial Use Standard* for a portable operating system environment based on Unix (POSIX) is currently under review for approval as a full use standard.

The last 1003 meeting, sponsored by AT&T, was the setting for the formation of two new working groups: the 1003.2 to consider the shell and tools facilities and the 1003.3 to address the issue of verification test specifications. All three groups will hold their next meeting in Palo Alto, California, September 17-19. Hosts will be Amdahl, Hewlett-Packard, and Sun Micro Systems.

Persons interested in participating in the IEEE 1003 efforts should contact Jim Isaak at Charles River Data Systems, Inc., 983 Concord Street, Framingham, MA. Copies of the trial use standard can be obtained for $19.95 from the IEEE Computer Society at (714) 821-8380. Ask for book no. 967.

JPL switches to micros to plan for Galileo flight

Microcomputers are doing their share in supporting the search of the universe for scientific data, according to an announcement made by Jet Propulsion Laboratory. JPL, a division of the California Institute of Technology in Pasadena, California, has the assignment of exploring the solar system with unmanned spacecraft such as Voyager and Galileo under a contract with the National Aeronautics and Space Administration.

Microcomputers were chosen by JPL recently to plan and schedule the handling of the vast amount of data involved in Galileo's collection of information about the weather, radio emissions, and magnetic field on Jupiter. Previously, JPL's Deep Space Network team had used a mainframe computer for project management and scheduling. But, the lab found it needed to cut costs and have more flexibility and convenience in its work.

Microcomputers and their software were a natural choice to accomplish both requirements, according to Hank Bell, a member of the JPL technical staff supporting the Galileo flight. "By moving project planning from mainframe to microcomputers, we have cut costs significantly," reported Bell. "And the software is much easier for our technical staff to work with."

JPL says the complex Galileo project is ready to launch as soon as the present ban on space exploration ends, with one exception. Because the spacecraft has a nuclear power source, a decision was made after the Challenger accident to develop a new shield for Galileo. Once the shield is perfected, Galileo will be ready to launch.

It appears that earth will also be ready to receive Galileo's reports when the time comes—and microcomputers will help.

Will optical computers surpass electronic computers?

Major international and US research programs into the development of optical computers and interconnects in VLSI are leading experts to predict an annual market for optical computers of over $1 billion by the year 2000. The possibility exists that within a few years an electronic minicomputer with an optical array processor will equal today's supercomputers in speed for selected problems, while offering cost, size, weight, power consumption, and reliability improvements.

These forecasts are featured in a SEAI Technical Publications report on the research presently being conducted by groups such as The European Joint Optical Bistability Program, Japan's Optical Computer Group, the US Defense Advanced Research Projects Agency, and the National Science Foundation; companies such as AT&T Bell Laboratories, Texas Instruments, Harris Corporation, Motorola, TRW, and Hughes; and a large number of universities.

While not yet to the commercial stage, optical computer technology is reasonably well developed and analog and image processing prototype systems have been demonstrated.

Further information about optical computing is available in the $680, 330-page *Optical Computers: The Next Frontier in Computing*, which is published by SEAI, PO Box 590, Madison, GA 30650; (404) 342-9638.

Graduate program announced

The Semiconductor Research Corporation, a consortium representing 63 US electronic firms, has announced plans for its first Graduate Fellowship Program to foster the growth of qualified personnel in the US semiconductor industry.

The program, to commence in the fall of 1986, will provide 20 selected individuals with full tuition and fee support for up to four years of PhD graduate study in the field of microelectronics. A monthly stipend and an unrestricted grant of $2000 to the university department in which the student is enrolled are major components of the program.

The program is part of the consortium's effort to promote a partnership with leading universities to expand the capabilities and sustain the excellence of the semiconductor industry within the United States.

NBS develops physical layer protocol test

The National Bureau of Standards Institute for Computer Sciences and Technology is developing a test method to ensure that modem and headend equipment from different manufacturers can communicate. The test method uses an IEEE standard, the 802.4 Broadband Token-Passing Bus physical layer protocol.

By transmitting and receiving a variety of test frames, modems from various manufacturers will be tested against headend equipment from other manufacturers. The test methods will make it possible for vendors to know whether the complex "protocols," which make possible multivendor computer-to-computer communication, are being implemented correctly.

US gate array market analyzed

Almost three-quarters of all surveyed design engineers, managers, and vice presidents plan to use gate arrays, 69 percent want special functions, and 19 percent will pay over $50,000 for NREs. These findings are included in a Secus International report of its survey made of the US market to assist manufacturers in gate array product definition.

The 250-page report, *Gate Array Usage/Requirements: 1986,* also offers data profiles of the over 500 respondents and analyzes packaging, pricing, and problems with delivery and quality by vendor name.

For copies of the $975 report contact Secus International, 1011 Elkton Drive, Colorado Springs, CO 80907; (303) 599-0405.

DSP guides

Texas Instruments and Motorola Inc. are offering guides to their DSP56000 and TMS320 digital signal processors.

The TI reference, *Digital Signal Processing Applications with the TMS320 Family,* discusses common DSP routines such as fast Fourier transforms and typical applications such as telecommunications and computers and peripherals, with application-specific source codes included. To order the free guidebook, call the company's Customer Response Center at (800) 232-3200 and request a copy of SPRA012.

Motorola's DSP user's manual, *DSP56000UM/AD,* provides chapters on device signal descriptions, chip architecture, data organization and addressing modes, and the instruction set, among others. It sells for $8.65 from the Motorola Literature Distribution Center, PO Box 20924, Phoenix, AZ 85036. ATechnical Summary is also available at no charge.

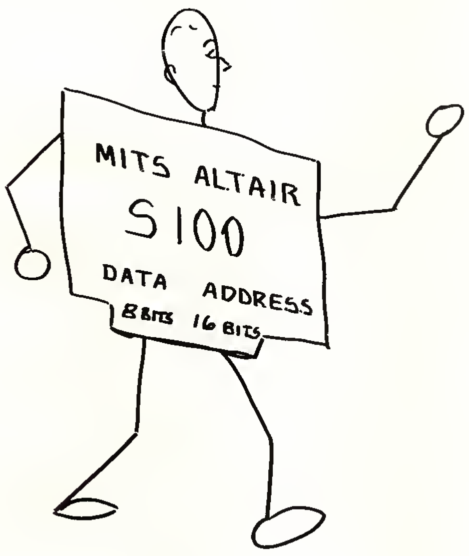
Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

High 186 Medium 187 Low 188

MicroStandards

Promises, promises, promises



Robert G. Stewart
Stewart Research Enterprises

Having reached the four-year limit for tenure on the editorial board, now in my last column I'll fulfill promises made to three editors-in-chief of *IEEE Micro*: Dick Jaeger, Peter Rony, and Jim Farrell. Dick and Peter participated in the creation and initiation of this magazine. Then each spent two years as EIC, which they tell me is a very demanding, unpaid, half-time labor of love. I promised them at the start of *Micro* not to criticize the magazine for a period of time. A sufficient period has now passed to allow a reasonable review of *Micro*'s actual performance.

With sheets

First, a slight digression: a favorite story of mine is *Animal Farm* by George Orwell. After the oppressed animals revolt and oust Farmer Jones, they prepare an animal's Bill of Rights, which they post in big letters on the side of the barn. One of the rights says that "No animal shall sleep in a bed." As the years go by, the pigs become the dominant animals on the farm and find sleeping in the old beds of Farmer Jones very comfortable. The animal's Bill of Rights was slightly altered to read, "No animal shall sleep in a bed with sheets." However, no animal could remember that this change was ever approved, nor how the Bill of Rights was altered. The actual course of events sometimes differs from initial ideals and goals; sometimes for the better, sometimes not.

A true letter

After the Computer Society's Governing Board passed unanimously my motion to initiate *IEEE Micro* on February 29, 1980, a statement of need and coverage for the new journal was written and sent to hard-working True Seaborn, editor and publisher of the CS magazines, at his request. To compare those early ideas with what is now history, let's look at that six-year-old letter to True.

"Potential authors of microprocessor articles have told me of their dilemma in not having a suitable IEEE journal in which to publish their work. The *Transactions on Electron Devices* and the *Journal of Solid-State Circuits* cover only some aspects of devices, and have not proven to be the journals of record for articles on microprocessor architecture, nor on applications of them. Some authors have told me that they had to alter their papers by emphasizing the chip layout or digital logic aspects in order to qualify for publications in those journals or to present papers at conferences sponsored by those groups!

"*Micro* should be a magazine in which to publish the following types of articles:

• Articles of record detailing architectural, processing, and system aspects of microprocessors and microcomputers,

• Hardware implementations detailing

chips and boards and buses used in actual systems,

- Software for applications including program listings up to several pages in length if appropriate,
- Tutorials and introductory articles,
- Draft standards, if appropriate.

"In addition, usual magazine features should be included such as new product announcements, letters to the editor, conference reports, meeting announcements, Computer Society, IEEE, and other organizational or outside activities, etc. Advertising would be beneficial to provide good current product awareness."

Hopes fulfilled???

Well, Dick and Peter, have those initial hopes been fulfilled? I'd have to answer, "Mostly yes!" But one topic has not been covered well: the processing of microprocessors. As a solid-state physicist, I've always felt that everyone should be knowledgeable about the art of taking a silicon wafer and creating working devices and circuits upon it. So I hope that you, the readers and the editorial board of *Micro*, foster the submission of papers dealing with chip design and processing.

Another deficiency is the paucity of advertisements in *Micro*. A reasonable number of ads is acceptable since they do help bring needed products to the reader's attention. I don't pretend to know the answer for the deficiency of ads, but maybe the readers can jog the marketing groups in their companies to use this media more effectively and also to be responsive to the bingo card for the ads that do appear.

What has *Micro* done well? Most of the other initial goals have been rather nicely fulfilled. The authors of manuscripts really determine the overall mix of article contents. The editorial board can foster special issues and solicit particularly appealing articles. Actually I think *Micro*'s EICs have done a remarkably good job, particularly remembering that they are unpaid volunteers, not paid professionals. Some papers have been truly outstanding examples of making a difficult, abstract subject intelligible to the nonexpert. The recent articles by Daniel Mange on microcoding bit-slice processors in the February and April issues were of that quality. The efforts of manuscript referees and the managing editor are crucial in making articles understandable. This you don't realize unless you are on the editorial board and compare the initial papers with the published documents. The extensive use of tutorial "sidebars" plus sufficient page count for each article to allow depth and thoroughness has made *Micro* good.

Promise three

The promise to Jim Farrell was to write a brief introduction to the following document, "A Historical ? View of IEEE Standards During the Great Bus Wars." The document was prepared at Jim's request for a session on 32-bit buses at the Mini-Micro Conference held in Boston in May of this year. The format is a wee bit different. Now I've been reading IEEE journals for at least 30 years, and I've come to the conclusion, perhaps wrongly, that the usual readership of an article consists of the author, his wife or girlfriend, his boss, the associate editor and reviewers of the manuscript, and sometimes additional individuals: mother, aunt, et al. The style traditionally used in professional literature seems to have been inherited from European college professors dating from the 1600's on. They sometimes wrote in Latin and/or conundrums just to make sure the reader really had to sweat to decode what the author meant. Some with pride went out of their way to eliminate all figures. That's particularly inspiring in the case of a text on mechanics. Fortunately they've stopped using Latin. But, unfortunately, many seem to have switched to Greek.

Today we are the victims of our current technology—microcomputers and word processors; they're great for text but poor for figures. Thus many articles or books are woefully devoid or deficient in artwork. I hope this will change as megabyte picture memories become plentiful. Typically now we deal with words not pictures in our technical articles. Yet when an engineer conceives a circuit, he takes a pad of paper and draws a schematic, as compared to writing out an algorithm.

As I sat down in February to write the article for Mini-Micro that Jim Farrell had solicited last fall, it seemed to me that history can be pretty dry reading, so the only ones likely to read it would be Jim, myself, my girlfriend Cloria, kindly relatives, and possibly the individuals directly mentioned in the document. In the back of my mind was a marvelous book, *Approximations for Digital Computers* by Cecil Hastings, which teaches the art of developing approximation functions to data using artwork, figures, and drawings. The pictorial impressions leave lasting concepts embedded in one's memory. Maybe the format chosen will foster a higher percentage of readership than the more traditional manuscript. (Use the bingo card to let Jim know.)

Lest you think that the chosen format was easy to prepare, I can tell you that three weeks were required to make the drawings. They started as "stick drawings" but gradually changed to more realistic efforts to portray individuals. Old copies of *Computer* and *IEEE Micro* were helpful in providing pictures.

Numbers

The development of standards is frowned upon by many in the computer profession. "Too early" and "not needed" are typical arguments. A favorite question of mine is to ask some college professor what MOV A,B means. They usually don't care that the meaning is currently ambiguous and manufacturer dependent. Neither do the manufacturers seem to care. Another point: Individuals do not become famous for developing standards. I think this is due to the fact that standards are known by a number like 488 or 802, whereas the usual publications or books are known by their authors' names. So authors become famous, while working group members do not. I hope the following history helps bring credit to the individuals mentioned therein.

Closets

What is the motivation to develop standards? Why is it worth doing? For me, the biggest problems have arisen in making equipment and/or software from different vendors work properly together. The design engineers in the companies seemingly consider themselves gurus and everyone else either idiots or nonexistent. They design in a closet. It is sometimes only years later that their sins are recognized. Frequently the company no longer exists. Ever try to get schematics for the hardware you've bought? "What, us release the fruit of our genius to the competition?" is the apparent attitude.

"Our equipment doesn't fail, so you won't have to repair it. You dont need schematics." Compare that attitude with the beautiful maintenance documents Sony and other Japanese firms have available. Perhaps it is another example of the great American industrial death wish.

Sour grapes? Yes. Do standards really help? Sometimes. Is it worth the effort to develop them? Probably yes.

The battles within the Computer Society and the IEEE over standards have been ferocious. *Computer* magazine has not published a single draft standard for five years. Thus you see the power of editors-in-chief.

To make clear the organizational structure involved in the following history, I include Figure 1 to show the standards development hierarchy within the Computer Society. After a draft is voted out by the working group, and approved by the management committee (here the MSC), it is balloted by the "sponsor," which is usually a Technical Committee of the Computer Society. After approval by the sponsor, it is forwarded to the IEEE Standards Board for final adoption as an IEEE standard. The Standards Activity Board of the Computer Society secures funding from the Board of Governors, fosters initiation of new standards, and oversees the mechanics of standards development and coordination with other standards-making bodies.

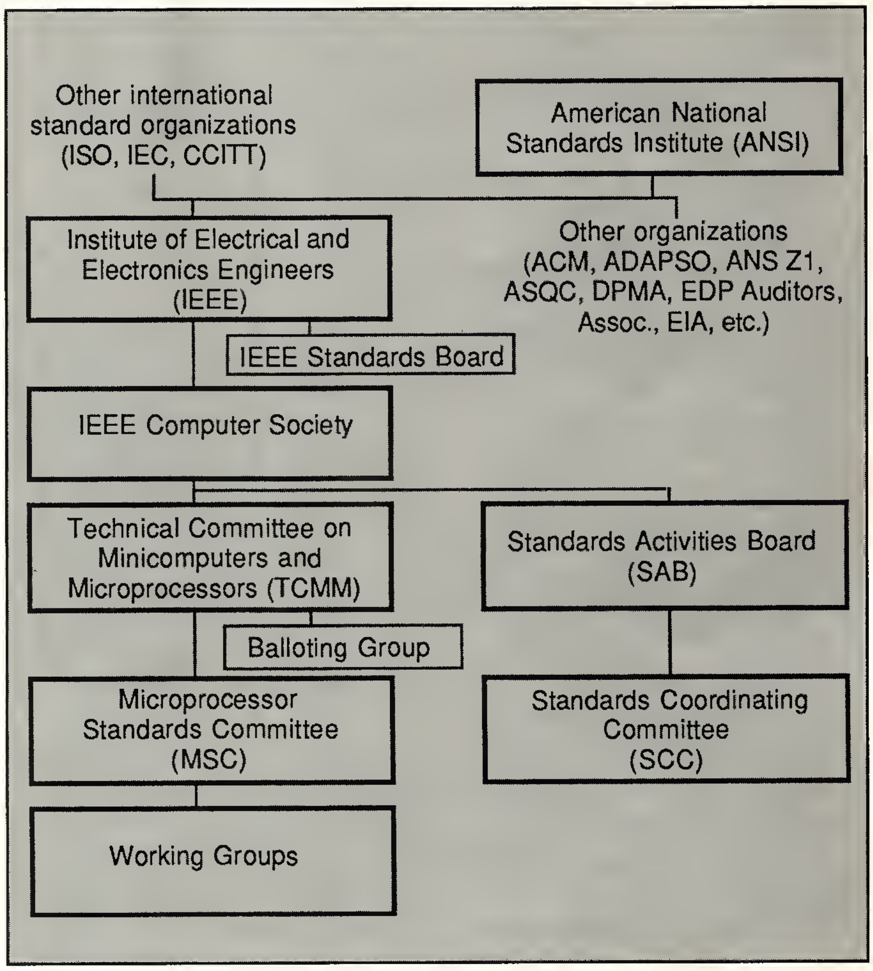

Other international standard organizations (ISO, IEC, CCITT)
American National Standards Institute (ANSI)
Institute of Electrical and Electronics Engineers (IEEE)
Other organizations (ACM, ADAPSO, ANS Z1, ASQC, DPMA, EDP Auditors, Assoc., EIA, etc.)
IEEE Standards Board
IEEE Computer Society
Technical Committee on Minicomputers and Microprocessors (TCMM)
Standards Activities Board (SAB)
Balloting Group
Microprocessor Standards Committee (MSC)
Standards Coordinating Committee (SCC)
Working Groups

Figure 1. CS standards-development environment.

Many of the stories recounted in the following history happened in the setting of Silicon Valley, as the beautiful Santa Clara Valley of California south of Palo Alto has become known, during the heyday of the microcomputer revolution. Tom Pittman alluded to the group developing microcomputer standards as the "Silicon Valley Mafia." That appellation was probably purely allegorical, but I deem it mandatory to hereby categorically deny any and all connections to the syndicate. In actuality, many individuals from all over the world devoted an incredible number of hours to these standards efforts. It's been a real honor to know people like Paul Borrill, Jim Cody, Dave Gustavson, Maris Graube, Velvel Kahan, Hubert Kirrmann, Tom Pittman, Matthew Taub, and the many others depicted in the pages that follow. God bless them all.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

High 189   Medium 190   Low 191

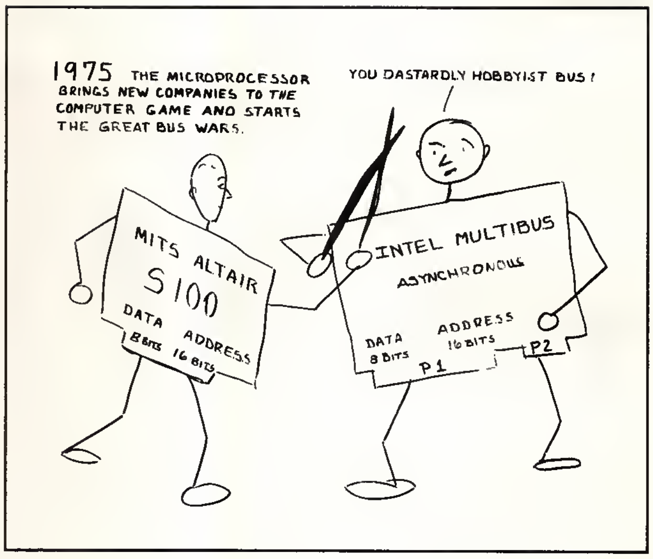
HISTORICAL ?
~~HYSTERICAL~~
~~HISTORICAL~~ VIEW OF
IEEE STANDARDS
DURING THE GREAT

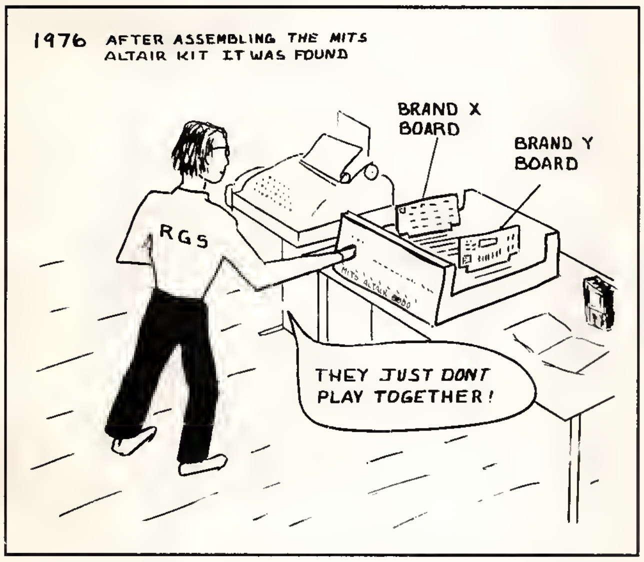
BUS WARS*

ROBERT G STEWART (RGS)

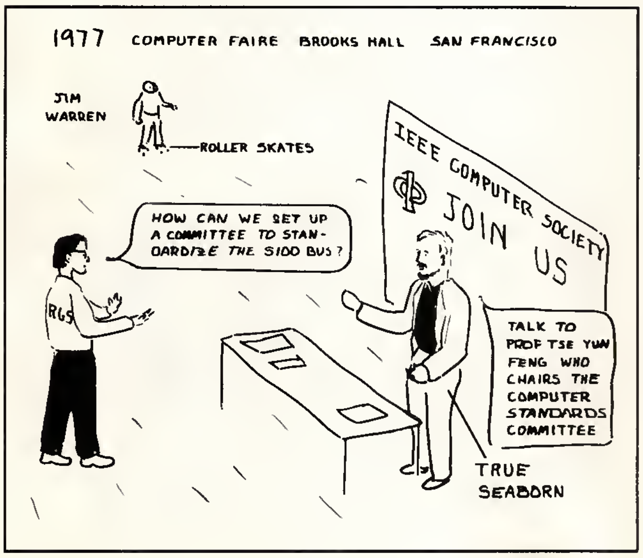
STEWART RESEARCH ENTERPRISES
LOS ALTOS, CALIFORNIA

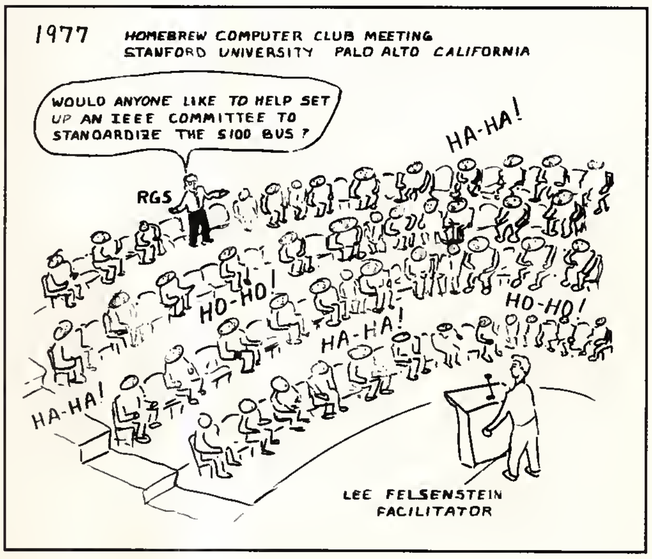
*Originally presented at Mini-Micro Northeast 86.

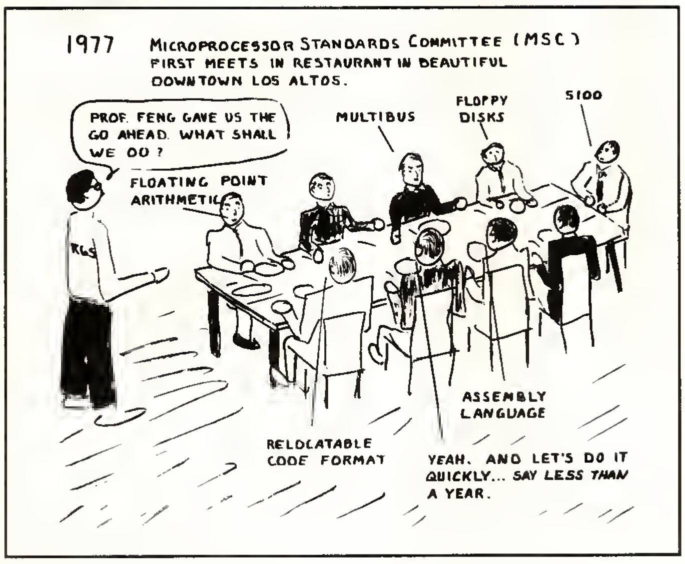

1975 THE MICROPROCESSOR BRINGS NEW COMPANIES TO THE COMPUTER GAME AND STARTS THE GREAT BUS WARS.
YOU DASTARDLY HOBBYIST BUS!
MITS ALTAIR S100 DATA 8 BITS ADDRESS 16 BITS
INTEL MULTIBUS ASYNCHRONOUS DATA 8 BITS ADDRESS 16 BITS P1 P2

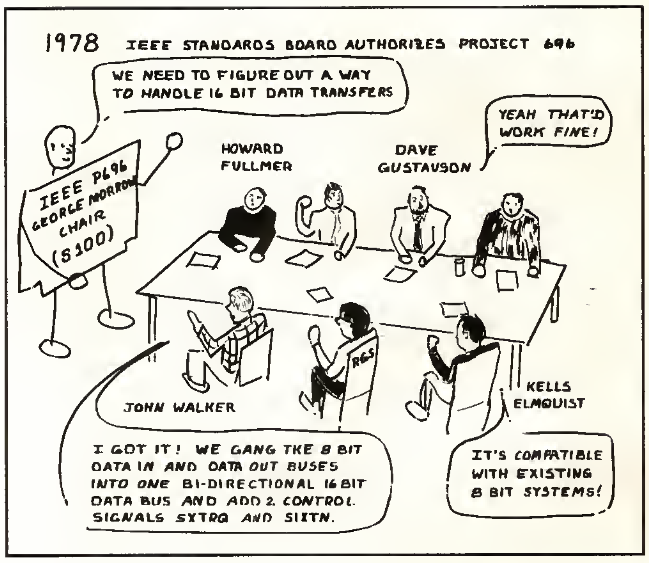

1976 AFTER ASSEMBLING THE MITS ALTAIR KIT IT WAS FOUND
BRAND X BOARD
BRAND Y BOARD
RGS
MITS ALTAIR 8800
THEY JUST DONT PLAY TOGETHER!

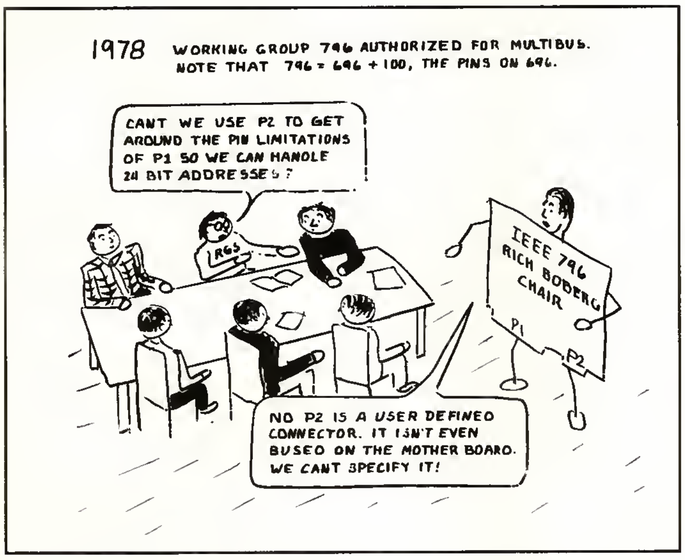
1977 HOMEBREW COMPUTER CLUB MEETING STANFORD UNIVERSITY PALO ALTO CALIFORNIA
WANNA BUY AN APPLE ?
THIS WILL CERTAINLY FLOP!
WHY A 6502 PROCESSOR ?
STEVE JOBS
RGS
STEVE WOZNIAK
IT WAS $2.00 CHEAPER.

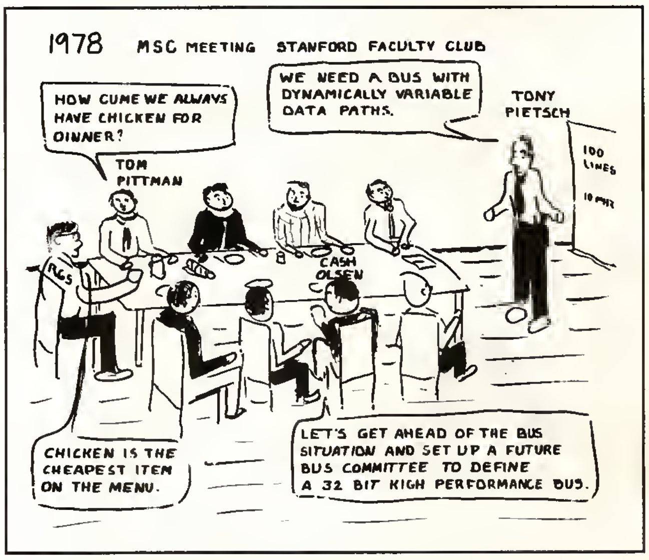

1977 COMPUTER FAIRE BROOKS HALL SAN FRANCISCO
JIM WARREN
ROLLER SKATES
HOW CAN WE SET UP A COMMITTEE TO STANDARDIZE THE S100 BUS?
RGS
IEEE COMPUTER SOCIETY JOIN US
TALK TO PROF TSE YUN FENG WHO CHAIRS THE COMPUTER STANDARDS COMMITTEE
TRUE SEABORN

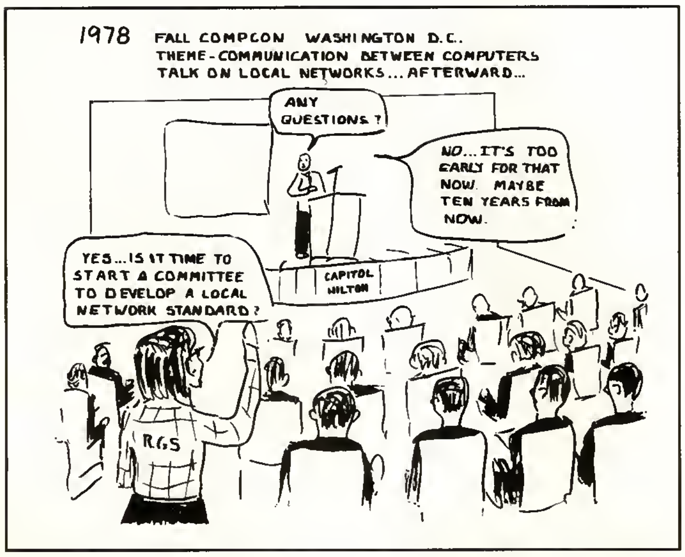

1977 HOMEBREW COMPUTER CLUB MEETING STANFORD UNIVERSITY PALO ALTO CALIFORNIA
WOULD ANYONE LIKE TO HELP SET UP AN IEEE COMMITTEE TO STANDARDIZE THE S100 BUS ?
HA-HA!
RGS
HO-HO!
HA-HA!
HO-HO!
HA-HA!
LEE FELSENSTEIN FACILITATOR

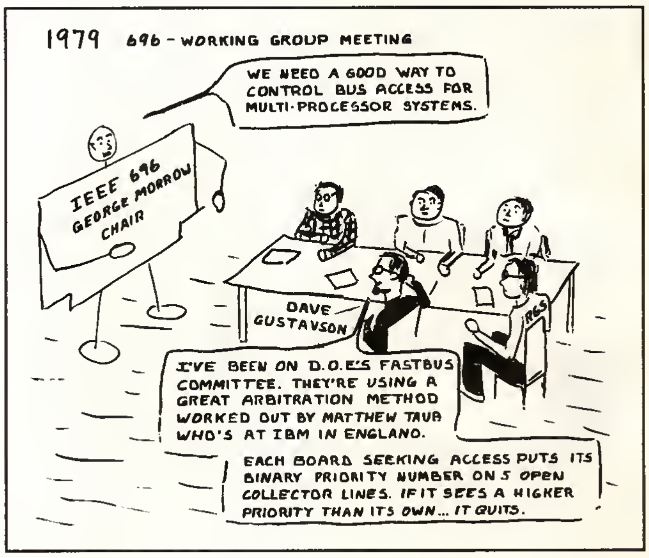

1977 MICROPROCESSOR STANDARDS COMMITTEE (MSC) FIRST MEETS IN RESTAURANT IN BEAUTIFUL DOWNTOWN LOS ALTOS.
PROF. FENG GAVE US THE GO AHEAD. WHAT SHALL WE DO?
FLOATING POINT ARITHMETIC
MULTIBUS
FLOPPY DISKS
S100
RGS
ASSEMBLY LANGUAGE
RELOCATABLE CODE FORMAT
YEAH. AND LET'S DO IT QUICKLY... SAY LESS THAN A YEAR.

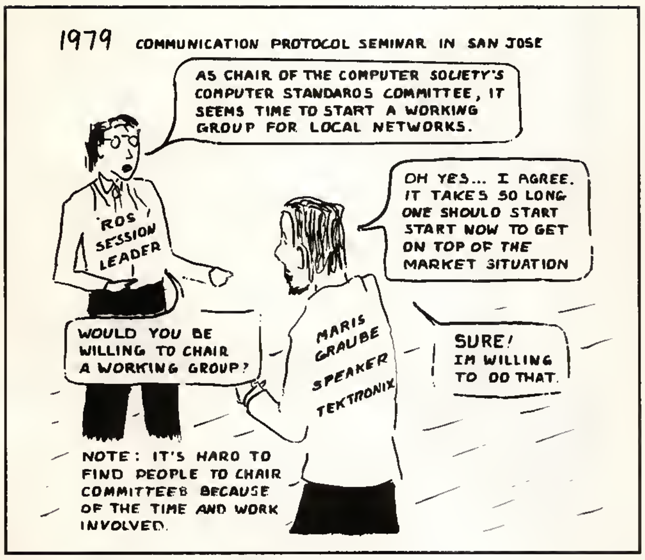

1978 IEEE STANDARDS BOARD AUTHORIZES PROJECT 696
WE NEED TO FIGURE OUT A WAY TO HANDLE 16 BIT DATA TRANSFERS
IEEE P696 GEORGE MORROW CHAIR (S100)
HOWARD FULLMER
DAVE GUSTAVSON
YEAH THAT'D WORK FINE!
RGS
JOHN WALKER
KELLS ELMQUIST
I GOT IT! WE GANG THE 8 BIT DATA IN AND DATA OUT BUSES INTO ONE BI-DIRECTIONAL 16 BIT DATA BUS AND ADD 2 CONTROL SIGNALS SXTRQ AND SIXTN.
IT'S COMPATIBLE WITH EXISTING 8 BIT SYSTEMS!

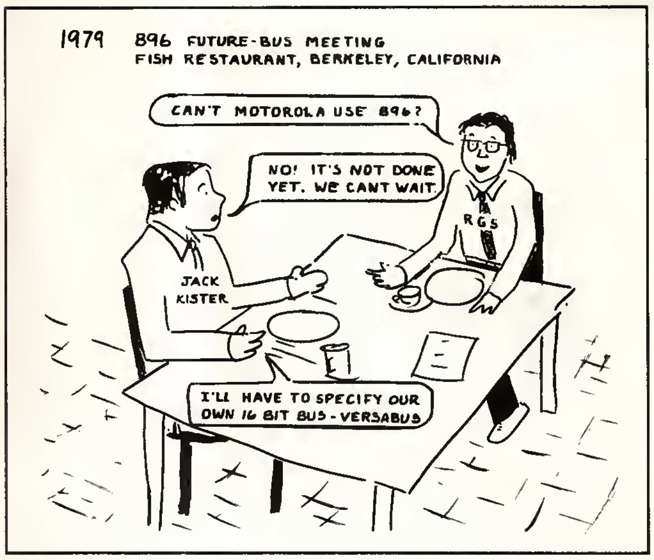

1978 WORKING GROUP 796 AUTHORIZED FOR MULTIBUS. NOTE THAT 796 = 696 + 100, THE PINS ON 696.
CAN'T WE USE P2 TO GET AROUND THE PIN LIMITATIONS OF P1 SO WE CAN HANDLE 24 BIT ADDRESSES?
RGS
IEEE 796 RICH BOBERG CHAIR
P1
P2
NO P2 IS A USER DEFINED CONNECTOR. IT ISN'T EVEN BUSED ON THE MOTHER BOARD. WE CANT SPECIFY IT!

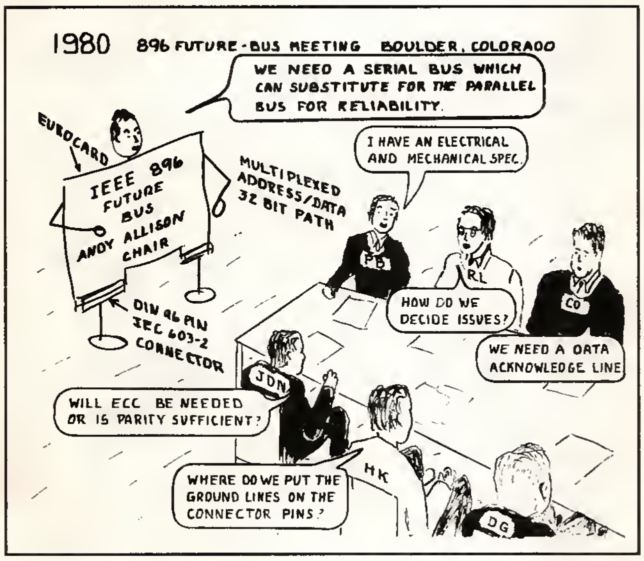

1978 MSC MEETING STANFORD FACULTY CLUB
HOW COME WE ALWAYS HAVE CHICKEN FOR DINNER?
TOM PITTMAN
WE NEED A BUS WITH DYNAMICALLY VARIABLE DATA PATHS.
TONY PIETSCH
100 LINES
10 MHz
RGS
CASH OLSEN
CHICKEN IS THE CHEAPEST ITEM ON THE MENU.
LET'S GET AHEAD OF THE BUS SITUATION AND SET UP A FUTURE BUS COMMITTEE TO DEFINE A 32 BIT HIGH PERFORMANCE BUS.

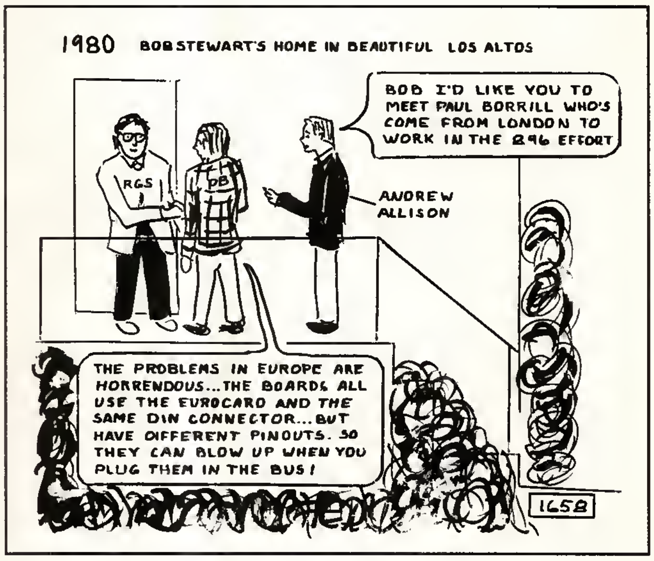

1978 FALL COMPCON WASHINGTON D.C. THEME-COMMUNICATION BETWEEN COMPUTERS TALK ON LOCAL NETWORKS... AFTERWARD...
ANY QUESTIONS?
NO... IT'S TOO EARLY FOR THAT NOW. MAYBE TEN YEARS FROM NOW.
YES... IS IT TIME TO START A COMMITTEE TO DEVELOP A LOCAL NETWORK STANDARD?
CAPITOL HILTON
RGS

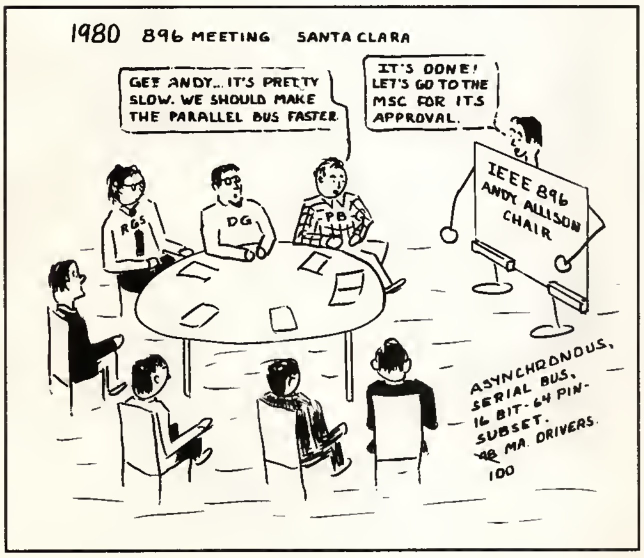

1979 696-WORKING GROUP MEETING
WE NEED A GOOD WAY TO CONTROL BUS ACCESS FOR MULTI-PROCESSOR SYSTEMS.
IEEE 696 GEORGE MORROW CHAIR
DAVE GUSTAVSON
RGS
I'VE BEEN ON D.O.E.'S FASTBUS COMMITTEE. THEY'RE USING A GREAT ARBITRATION METHOD WORKED OUT BY MATTHEW TAUB WHO'S AT IBM IN ENGLAND.
EACH BOARD SEEKING ACCESS PUTS ITS BINARY PRIORITY NUMBER ON 5 OPEN COLLECTOR LINES. IF IT SEES A HIGHER PRIORITY THAN ITS OWN... IT QUITS.

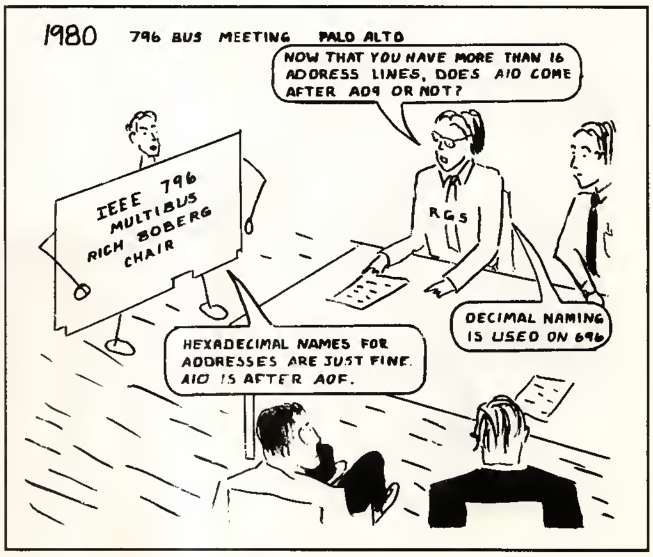

1979 COMMUNICATION PROTOCOL SEMINAR IN SAN JOSE
AS CHAIR OF THE COMPUTER SOCIETY'S COMPUTER STANDARDS COMMITTEE, IT SEEMS TIME TO START A WORKING GROUP FOR LOCAL NETWORKS.
RDS SESSION LEADER
OH YES... I AGREE. IT TAKES SO LONG ONE SHOULD START START NOW TO GET ON TOP OF THE MARKET SITUATION
WOULD YOU BE WILLING TO CHAIR A WORKING GROUP?
MARIS GRAUBE SPEAKER TEKTRONIX
SURE! IM WILLING TO DO THAT.
NOTE: IT'S HARD TO FIND PEOPLE TO CHAIR COMMITTEES BECAUSE OF THE TIME AND WORK INVOLVED.

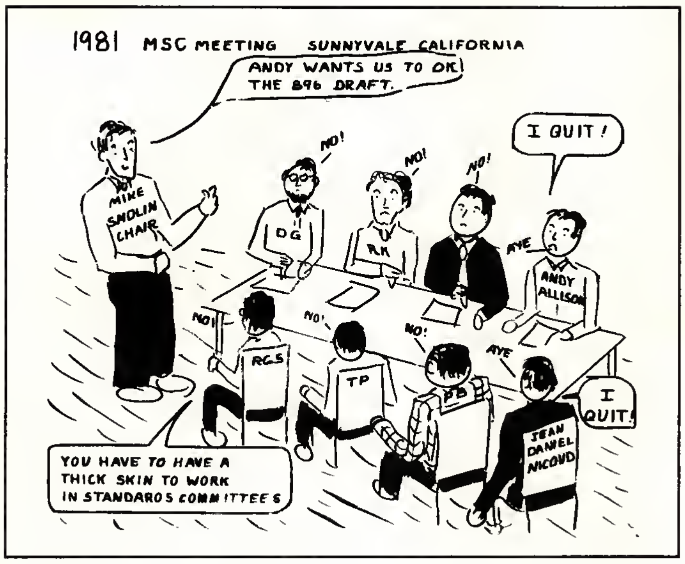

1979 896 FUTURE-BUS MEETING FISH RESTAURANT, BERKELEY, CALIFORNIA
CAN'T MOTOROLA USE 896?
NO! IT'S NOT DONE YET. WE CANT WAIT.
RGS
JACK KISTER
I'LL HAVE TO SPECIFY OUR OWN 16 BIT BUS - VERSABUS

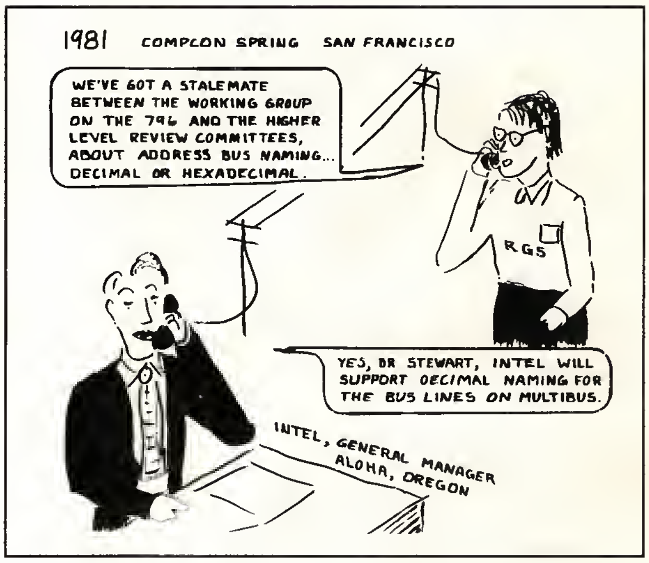

1980 896 FUTURE-BUS MEETING BOULDER, COLORADO
WE NEED A SERIAL BUS WHICH CAN SUBSTITUTE FOR THE PARALLEL BUS FOR RELIABILITY.
EUROCARD
IEEE 896 FUTURE BUS ANDY ALLISON CHAIR
MULTIPLEXED ADDRESS/DATA 32 BIT PATH
I HAVE AN ELECTRICAL AND MECHANICAL SPEC.
PB
RL
CO
HOW DO WE DECIDE ISSUES?
DIN 96 PIN IEC 603-2 CONNECTOR
WE NEED A DATA ACKNOWLEDGE LINE
JDN
WILL ECC BE NEEDED OR IS PARITY SUFFICIENT?
HK
WHERE DO WE PUT THE GROUND LINES ON THE CONNECTOR PINS?
DG

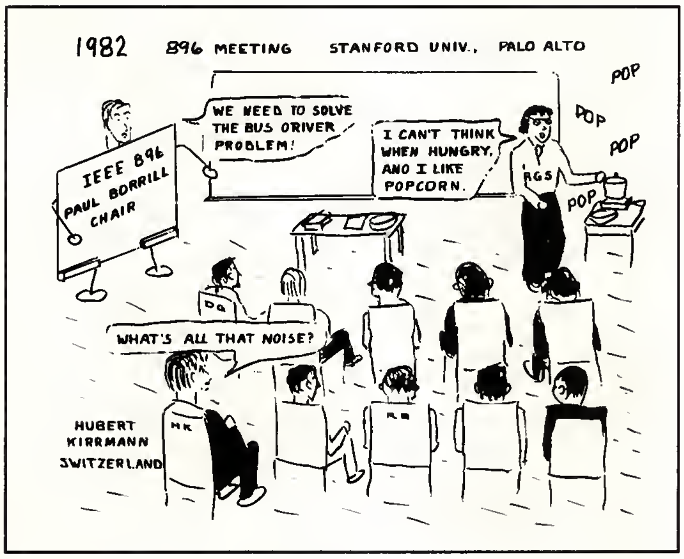

1980 BOB STEWART'S HOME IN BEAUTIFUL LOS ALTOS
BOB I'D LIKE YOU TO MEET PAUL BORRILL WHO'S COME FROM LONDON TO WORK IN THE 896 EFFORT
RGS
PB
ANDREW ALLISON
THE PROBLEMS IN EUROPE ARE HORRENDOUS...THE BOARDS ALL USE THE EUROCARD AND THE SAME DIN CONNECTOR...BUT HAVE DIFFERENT PINOUTS. SO THEY CAN BLOW UP WHEN YOU PLUG THEM IN THE BUS!
1658

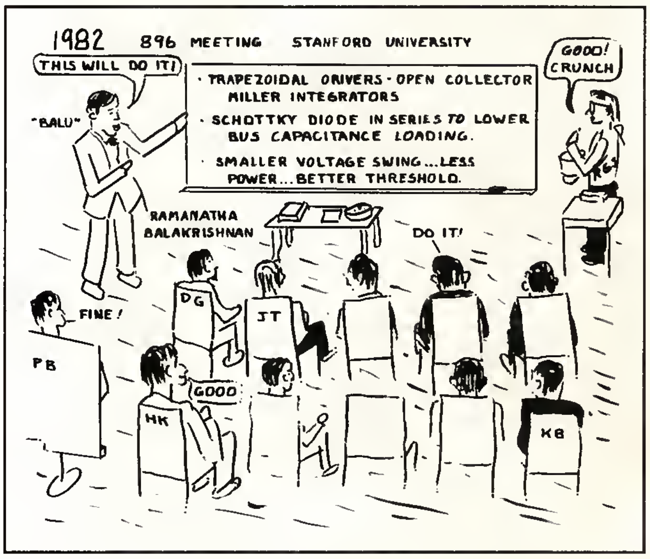

1980 896 MEETING SANTA CLARA
GEE ANDY... IT'S PRETTY SLOW. WE SHOULD MAKE THE PARALLEL BUS FASTER.
IT'S DONE! LET'S GO TO THE MSC FOR ITS APPROVAL.
RGS
DG
PB
IEEE 896 ANDY ALLISON CHAIR
ASYNCHRONOUS, SERIAL BUS, 16 BIT-64 PIN-SUBSET. ~~48~~ MA. DRIVERS.
100

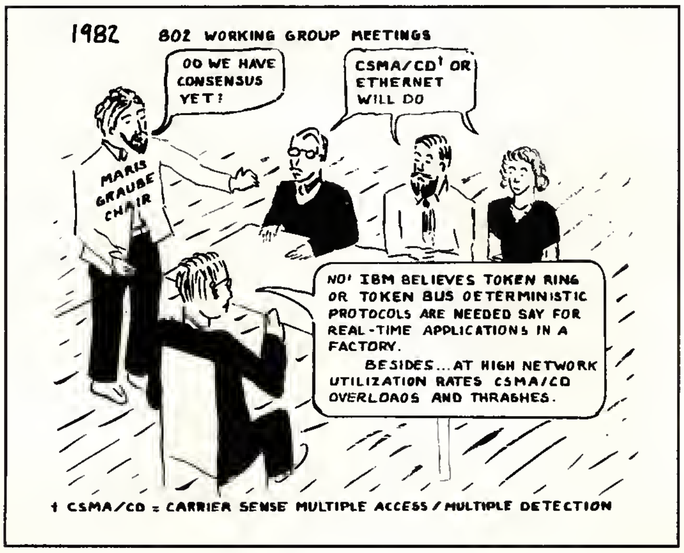

1980 796 BUS MEETING PALO ALTO
NOW THAT YOU HAVE MORE THAN 16 ADDRESS LINES, DOES A10 COME AFTER A09 OR NOT?
IEEE 796 MULTIBUS RICH BOBERG CHAIR
RGS
DECIMAL NAMING IS USED ON 696
HEXADECIMAL NAMES FOR ADDRESSES ARE JUST FINE. A10 IS AFTER A0F.

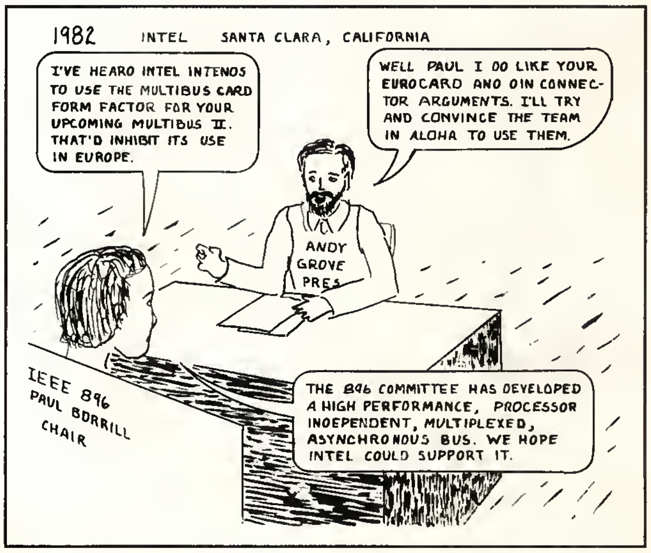

1981 MSC MEETING SUNNYVALE CALIFORNIA
ANDY WANTS US TO OK THE 896 DRAFT.
MIKE SMOLIN CHAIR
NO!
NO!
NO!
I QUIT!
AYE
DG
RK
ANDY ALLISON
NO!
NO!
NO!
AYE
RGS
TP
PB
I QUIT!
JEAN DANIEL NICOUD
YOU HAVE TO HAVE A THICK SKIN TO WORK IN STANDARDS COMMITTEES

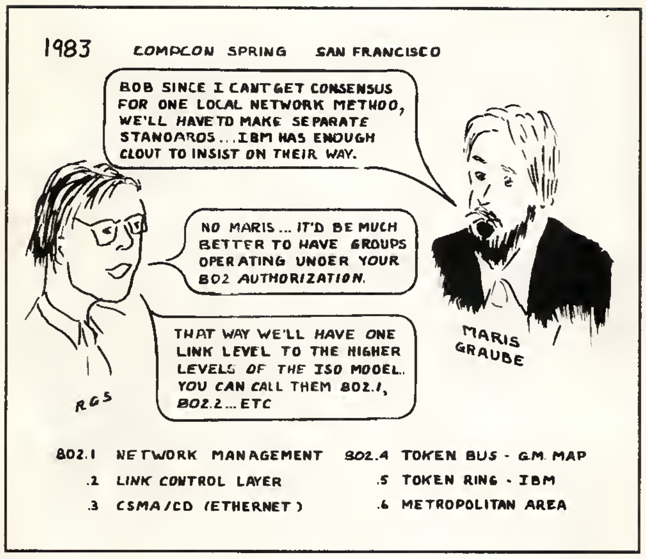

1981 COMPCON SPRING SAN FRANCISCO
WE'VE GOT A STALEMATE BETWEEN THE WORKING GROUP ON THE 796 AND THE HIGHER LEVEL REVIEW COMMITTEES, ABOUT ADDRESS BUS NAMING... DECIMAL OR HEXADECIMAL.
RGS
YES, DR STEWART, INTEL WILL SUPPORT DECIMAL NAMING FOR THE BUS LINES ON MULTIBUS.
INTEL, GENERAL MANAGER ALOHA, OREGON

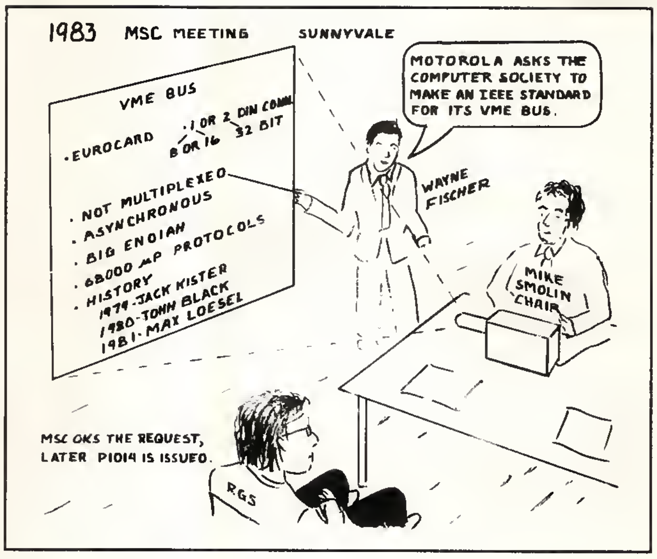

1982 896 MEETING STANFORD UNIV., PALO ALTO
WE NEED TO SOLVE THE BUS DRIVER PROBLEM!
IEEE 896 PAUL BORRILL CHAIR
I CAN'T THINK WHEN HUNGRY, AND I LIKE POPCORN.
RGS
POP
POP
POP
POP
DG
WHAT'S ALL THAT NOISE?
HUBERT KIRRMANN SWITZERLAND
HK
RB

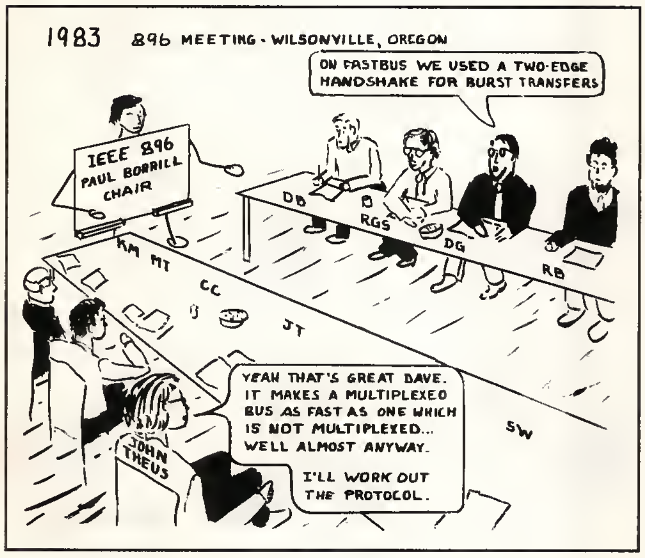

1982 896 MEETING STANFORD UNIVERSITY
THIS WILL DO IT!
"BALU"
· TRAPEZOIDAL DRIVERS - OPEN COLLECTOR MILLER INTEGRATORS
· SCHOTTKY DIODE IN SERIES TO LOWER BUS CAPACITANCE LOADING.
· SMALLER VOLTAGE SWING... LESS POWER... BETTER THRESHOLD.
GOOD! CRUNCH
RGS
RAMANATHA BALAKRISHNAN
DO IT!
FINE!
PB
DG
JT
GOOD
HK
KB

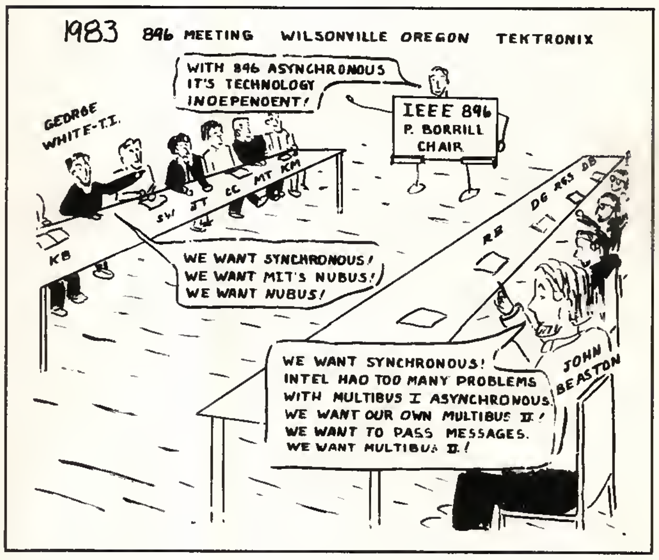

1982 802 WORKING GROUP MEETINGS
DO WE HAVE CONSENSUS YET?
CSMA/CD† OR ETHERNET WILL DO
MARIS GRAUBE CHAIR
NO! IBM BELIEVES TOKEN RING OR TOKEN BUS DETERMINISTIC PROTOCOLS ARE NEEDED SAY FOR REAL-TIME APPLICATIONS IN A FACTORY. BESIDES... AT HIGH NETWORK UTILIZATION RATES CSMA/CD OVERLOADS AND THRASHES.
† CSMA/CD = CARRIER SENSE MULTIPLE ACCESS / MULTIPLE DETECTION


1982 INTEL SANTA CLARA, CALIFORNIA
I'VE HEARD INTEL INTENDS TO USE THE MULTIBUS CARD FORM FACTOR FOR YOUR UPCOMING MULTIBUS II. THAT'D INHIBIT ITS USE IN EUROPE.
WELL PAUL I DO LIKE YOUR EUROCARD AND DIN CONNECTOR ARGUMENTS. I'LL TRY AND CONVINCE THE TEAM IN ALOHA TO USE THEM.
ANDY GROVE PRES
IEEE 896 PAUL BORRILL CHAIR
THE 896 COMMITTEE HAS DEVELOPED A HIGH PERFORMANCE, PROCESSOR INDEPENDENT, MULTIPLEXED, ASYNCHRONOUS BUS. WE HOPE INTEL COULD SUPPORT IT.


1983 COMPCON SPRING SAN FRANCISCO
BOB SINCE I CAN'T GET CONSENSUS FOR ONE LOCAL NETWORK METHOD, WE'LL HAVE TO MAKE SEPARATE STANDARDS... IBM HAS ENOUGH CLOUT TO INSIST ON THEIR WAY.
NO MARIS... IT'D BE MUCH BETTER TO HAVE GROUPS OPERATING UNDER YOUR 802 AUTHORIZATION.
THAT WAY WE'LL HAVE ONE LINK LEVEL TO THE HIGHER LEVELS OF THE ISO MODEL. YOU CAN CALL THEM 802.1, 802.2... ETC
MARIS GRAUBE
RGS
802.1 NETWORK MANAGEMENT
.2 LINK CONTROL LAYER
.3 CSMA/CD (ETHERNET)
802.4 TOKEN BUS - G.M. MAP
.5 TOKEN RING - IBM
.6 METROPOLITAN AREA

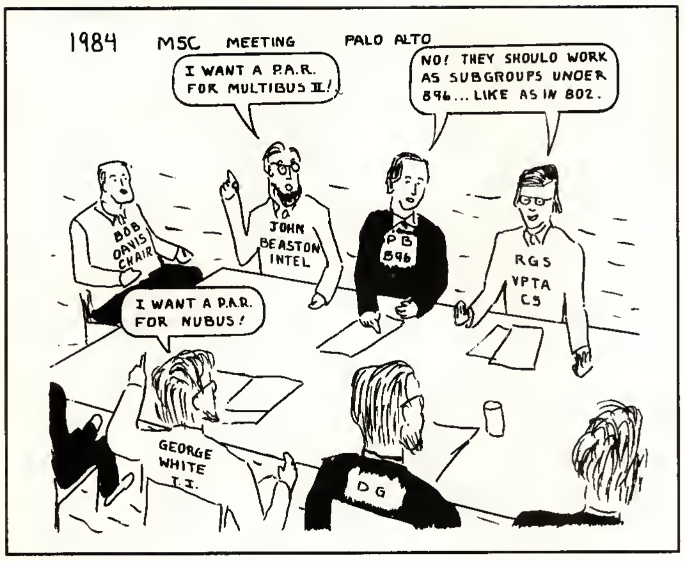

1983 MSC MEETING SUNNYVALE
VME BUS
• EUROCARD
• 1 OR 2 DIN CONN
8 OR 16
32 BIT
• NOT MULTIPLEXED
• ASYNCHRONOUS
• BIG ENDIAN
• 68000 μP PROTOCOLS
• HISTORY
1979-JACK KISTER
1980-JOHN BLACK
1981-MAX LOESEL
MOTOROLA ASKS THE COMPUTER SOCIETY TO MAKE AN IEEE STANDARD FOR ITS VME BUS.
WAYNE FISCHER
MIKE SMOLIN CHAIR
MSC OKS THE REQUEST, LATER P1014 IS ISSUED.
RGS

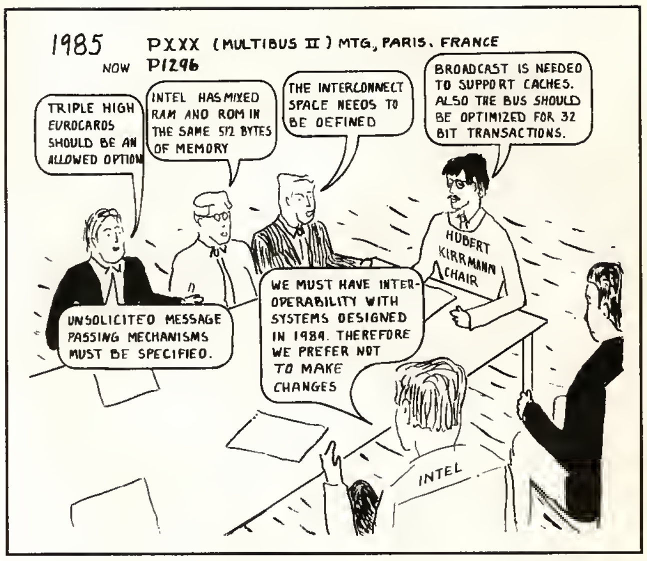

1983 896 MEETING - WILSONVILLE, OREGON
IEEE 896 PAUL BORRILL CHAIR
ON FASTBUS WE USED A TWO-EDGE HANDSHAKE FOR BURST TRANSFERS
DB
RGS
DG
RB
KM
MT
CC
JT
SW
JOHN THEUS
YEAH THAT'S GREAT DAVE. IT MAKES A MULTIPLEXED BUS AS FAST AS ONE WHICH IS NOT MULTIPLEXED... WELL ALMOST ANYWAY.
I'LL WORK OUT THE PROTOCOL.

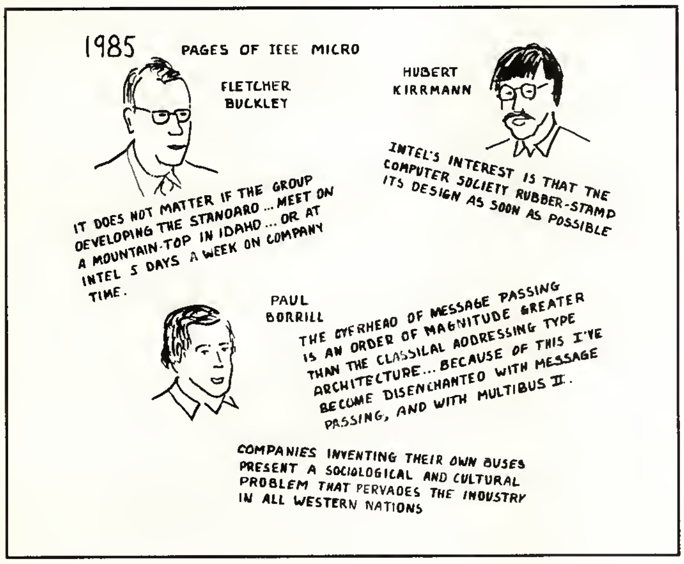

1983 896 MEETING WILSONVILLE OREGON TEKTRONIX
WITH 896 ASYNCHRONOUS IT'S TECHNOLOGY INDEPENDENT!
IEEE 896 P. BORRILL CHAIR
GEORGE WHITE-T.I.
KB
SW
JT
CC
MT
KM
RB
DG
RGS
DB
WE WANT SYNCHRONOUS! WE WANT MIT'S NUBUS! WE WANT NUBUS!
JOHN BEASTON
WE WANT SYNCHRONOUS! INTEL HAD TOO MANY PROBLEMS WITH MULTIBUS I ASYNCHRONOUS. WE WANT OUR OWN MULTIBUS II! WE WANT TO PASS MESSAGES. WE WANT MULTIBUS II!

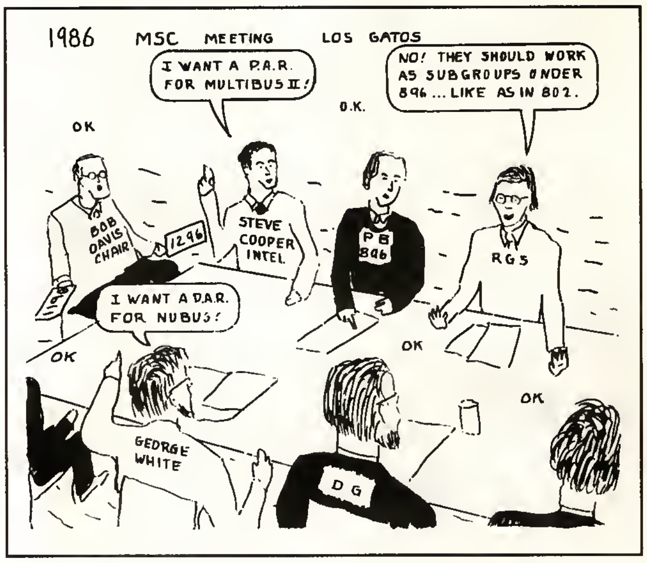
1983 MT. HOOD OREGON
THE TWEED JACKET IS REQUIRED! PROPRIETY ABOVE ALL, BOB.
PAUL DO YOU ENGLISHMAN ALWAYS SKI IN SHIRT, TIE, AND TWEED JACKET?
RGS

1983 MOUNT ST. HELENS, WASHINGTON TOUTLE RIVER CANYON
THAT ERUPTION THREE YEARS AGO LEFT NOTHING BUT GREY ASH, MUD, AND KNOCKED-DOWN TREES.
INCREDIBLE!
RGS
DR + MRS. MATTHEW TAUB
WORLD'S BIGGEST CLEAR-CUT!

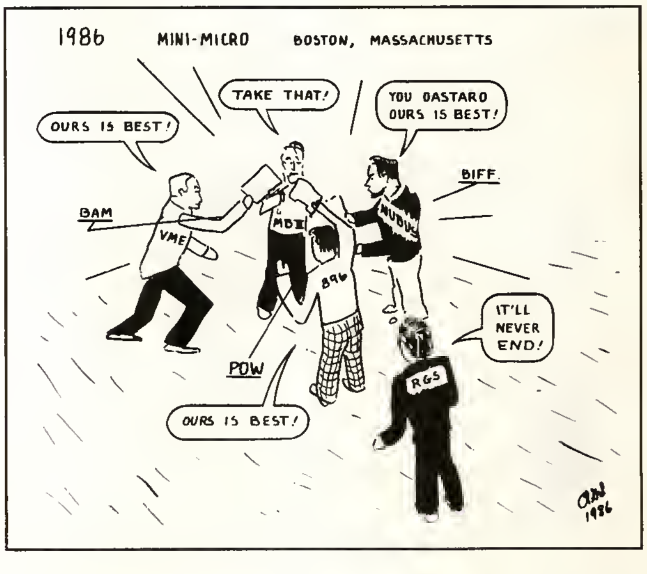

1984 MSC MEETING PALO ALTO
I WANT A P.A.R. FOR MULTIBUS II!
NO! THEY SHOULD WORK AS SUBGROUPS UNDER 896... LIKE AS IN 802.
BOB DAVIS CHAIR
JOHN BEASTON INTEL
PB 896
RGS VPTA CS
I WANT A P.A.R. FOR NUBUS!
GEORGE WHITE T.I.
DG


1985 PXXX (MULTIBUS II) MTG, PARIS, FRANCE
NOW P1296
TRIPLE HIGH EUROCARDS SHOULD BE AN ALLOWED OPTION
INTEL HAS MIXED RAM AND ROM IN THE SAME 512 BYTES OF MEMORY
THE INTERCONNECT SPACE NEEDS TO BE DEFINED
BROADCAST IS NEEDED TO SUPPORT CACHES. ALSO THE BUS SHOULD BE OPTIMIZED FOR 32 BIT TRANSACTIONS.
HUBERT KIRRMANN CHAIR
UNSOLICITED MESSAGE PASSING MECHANISMS MUST BE SPECIFIED.
WE MUST HAVE INTER-OPERABILITY WITH SYSTEMS DESIGNED IN 1984. THEREFORE WE PREFER NOT TO MAKE CHANGES
INTEL


1985 PAGES OF IEEE MICRO
FLETCHER BUCKLEY
IT DOES NOT MATTER IF THE GROUP DEVELOPING THE STANDARD... MEET ON A MOUNTAIN-TOP IN IDAHO... OR AT INTEL 5 DAYS A WEEK ON COMPANY TIME.
HUBERT KIRRMANN
INTEL'S INTEREST IS THAT THE COMPUTER SOCIETY RUBBER-STAMP ITS DESIGN AS SOON AS POSSIBLE
PAUL BORRILL
THE OVERHEAD OF MESSAGE PASSING IS AN ORDER OF MAGNITUDE GREATER THAN THE CLASSICAL ADDRESSING TYPE ARCHITECTURE... BECAUSE OF THIS I'VE BECOME DISENCHANTED WITH MESSAGE PASSING, AND WITH MULTIBUS II.
COMPANIES INVENTING THEIR OWN BUSES PRESENT A SOCIOLOGICAL AND CULTURAL PROBLEM THAT PERVADES THE INDUSTRY IN ALL WESTERN NATIONS


1986 MSC MEETING LOS GATOS
I WANT A P.A.R. FOR MULTIBUS II!
NO! THEY SHOULD WORK AS SUBGROUPS UNDER 896... LIKE AS IN 802.
OK
O.K.
BOB DAVIS CHAIR
1296
STEVE COOPER INTEL
PB 896
RGS
I WANT A P.A.R. FOR NUBUS!
OK
OK
OK
GEORGE WHITE
DG

1986 BUSCON SAN JOSE CONFERENCE CENTER
MOTOROLA VME IS BEST!
intel MB II IS BEST!
NATIONAL SEMICONDUCT
THE TOP TRACE IS 3897 FUTUREBUS DRIVER. THE LOWER IS TRI-STATE TTL
THE DIFFERENCE IS INCREDIBLE!
RGS
BALU
300 MHZ


1986 MINI-MICRO BOSTON, MASSACHUSETTS
OURS IS BEST!
TAKE THAT!
YOU BASTARD OURS IS BEST!
BIFF
BAM
VME
MB II
NUBUS
896
POW
OURS IS BEST!
IT'LL NEVER END!
RGS

MicroLaw continued from page 63

corner. Beta's Zener diode is rotated 90 degrees from the configuration of Alpha's. Probably, the Zener diode could have been placed farther away from its location in Alpha, but it is clear that this part of the chip was laid out and not cloned, so that the similarity between Alpha and Beta may well not be substantial for SCPA purposes.

Other circuitry

The remainder of the upper portion of each die is filled mainly with the other circuitry shown in the schematic, and there is little similarity. This part of the circuit does not seem to involve any colorable chip piracy.

The similarity between Alpha and Beta is more in the floor plan than in the structure or layout within modules. Where there is similarity within modules, it generally seems to be dictated by function. In all, the main similarity is that in laying out Beta B seems to have taken some aspects of the Alpha floor plan that are arbitrary and a matter of designer choice. In this regard, Beta follows Alpha "needlessly," or in what A's counsel could call "slavishly."

Was that action really needless? One consideration in second sourcing is satisfying the customer that the second chip is really compatible in form, fit, and function with the first chip. To some extent, unless the dies look alike, the customer may disbelieve the claim that the two chips are indeed compatible in form, fit, and function.

A further factor—that will probably tend to aid plaintiffs in infringement of mask work rights cases—is that many engineers do not believe in "reinventing the wheel." If one of several possible layouts has already been adopted by the first source, there is a tendency for the second-source designer not to change the layout unless there is a good reason to do so (that is, a functional one).

As indicated in the earlier part of this series (see *IEEE Micro*, June 1986, pp. 79-83), there are three separate criteria for establishing reverse engineering as a defense in an infringement of mask work rights case. First, the defendant must show records proving a substantial expenditure of its money, toil, study, and so on. In this case B, the manufacturer of Beta, was prepared to do so.

Second, the two layouts must not be substantially identical. This test applies to the die as a whole and also to substantial parts within it, such as the power output section. Some problems are raised for B and the Beta chip by B's "needless" duplications of Alpha's layout, which may have resulted from these factors:

- a desire to pacify the customer, who is concerned that the two chips are indeed compatible in form, fit, and function;
- an engineer's unwillingness to "reinvent the wheel"; and
- a rush to complete the layout and get the chip available for sale while there is still opportunity to become a second source.

The last point involves some questions about which data from B is unavailable, but the point is important. The legislative history of the SCPA indicates that one of the values it seeks to preserve is the innovator's head start, in that a second comer's cutting of corners by avoiding the innovator's design time and expense is considered misappropriation of the innovator's toil and investment.

For example, if A required six to eight months to lay the Alpha chip out and B needed only one to two months to lay the Beta chip out, it might suggest misappropriation to a jury or other fact finder. Hence, if that type of corner cutting to save time is the explanation for the "needless" duplications, it would point to chip piracy. On the other hand, if Beta's designers were very inefficient and took nearly as long to duplicate Alpha as it took Alpha's designers to lay Alpha out in the first place, it might suggest to a jury that Beta is the product of legitimate reverse engineering. There is a certain lack of logic to this, but that is how actual cases often seem to work.

The third and most difficult criterion to understand for reverse engineering is whether the second chip is an "original mask work." Is Beta different enough from Alpha to satisfy this mysterious test? Is Beta more than trivially different from Alpha? For example, has Beta independent merit as a design—in regard to points such as the improved temperature-compensation resistor design at the lower left? B, the manufacturer of Beta, may, of course, challenge Alpha as lacking any more independent merit as a design than Beta has. It is uncertain whether that argument would get B any place.

The questions are sufficiently close in the case of Alpha and Beta to make it an appropriate example to illustrate some of the factors involved in developing either side of a case of infringement of mask work rights, particularly areas in which expert testimony should be developed.

Case evaluation

Readers are invited to use the bingo card to give their opinion of whether B illicitly pirated or legitimately reverse engineered Alpha. Circle 101 for a clear case of piracy, 102 for definitely a case of reverse engineering, and 103 for too close to call.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

High 174 **Medium** 175 **Low** 176

New Products

Editor: Kenneth Majithia/IBM Corporation

Send announcements of new microcomputer/microprocessor products, and products for review, to Managing Editor, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.

Simulation speeded for 1M-bit memories

Logic Automation, Inc., is offering logic simulation models for 1M-bit memory devices and Motorola 68000 microprocessors. According to the company, its proprietary memory allocation technique uses a minimum amount of memory during simulation, thus decreasing the amount of system memory needed by two orders of magnitude.

SmartModels logic simulation models run on CAE workstations and include a design troubleshooting capability. The models perform all the functions of the parts themselves and are written at the behavioral level for fast performance.

Bus functional and full-functional models are available for 68000 simulation. Designers write a processor control file to execute bus cycles for hardware debugging. Software can be debugged with the full-functional model.

A 1M-bit EPROM SmartModel sells for $750 and a 1M-bit DRAM model for $950. Prices of the 680X0 SmartModels extend from $1500 to $6900. All are available from Logic Automation, 19545 NW Von Neumann Drive, PO Box 310, Beaverton, OR 97075; (503) 690-6900.

Reader Service Number 20

NCR adds cell generators to ASIC design system

NCR's VIGEN workstation-based system brings generators and compilers developed in the silicon compilation environment from Silicon Design Labs to Mentor Graphics Corp.'s IDEA engineering workstations running NCR's VLSI Design System.

The system designer can customize a VIGEN configurable cell by specifying parameters when incorporating the cell into an application-specific integrated circuit design. VIGEN allows the designer to specify and generate a desired logic block while still in NETED, the Mentor Graphics schematic capture system.

Medium-level configurable functions available include counters, multiplexers, and shift registers. Higher level function examples are EEPROM, RAM, and ROM memories and microprocessors and controllers.

VIGEN costs are about $15,000 per system; the price varies with the number of licensed generators/compilers. VIGEN is expected to be available to design centers and NCR beta site customers in 3Q 1986 and general release this winter.

Contact NCR Microelectronics, Public Relations, Dayton, OH 45479; (303) 226-9500.

Reader Service Number 21

Motorola announces a broadband interface controller

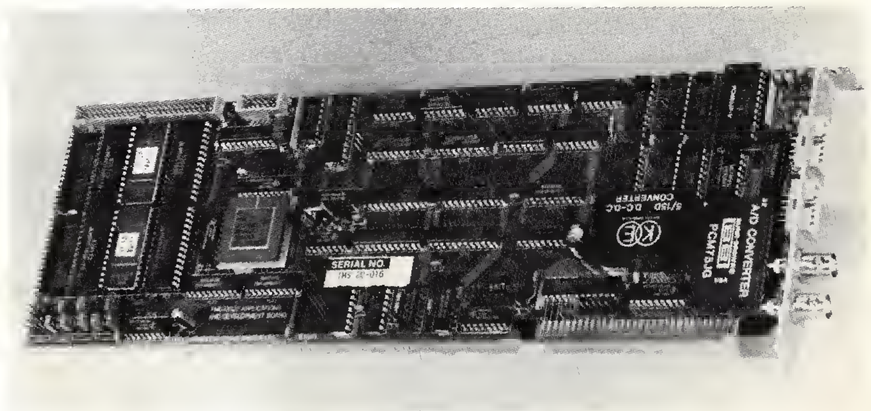
The MC68184 Broadband Interface Controller from Motorola Microprocessor Products Group supports General Motors Manufacturing Automation Protocol for real-time communications networking.

BIC, coupled with rf circuitry, makes up a broadband modem needed in each node of a MAP broadband network. The CMOS MC68184 BIC implements the digital portion of the IEEE 802.4 broadband physical layer of the ISO Open System Interconnection model for multivendor networking.

BIC supports high-speed data rates of 10M bps using a duo-binary modulation technique and connects to the MC68824 Token Bus Controller.

The 40-lead, dual-in-line plastic BIC sells for $40 in minimum quantities of 100. For more information contact a local Motorola Sales Office.

Reader Service Number 22



The TMS32020 Development System from Pacific Microcircuits Ltd. plugs into an IBM PC, XT, AT, or compatible. The system contains 32020-based hardware, 16-bit A/D and D/A converters, Monitor software, and Texas Instruments Macro Assembler/Linker. The board, software, and documentation sell for $2595. Contact PML at 240 H Street, Blaine, WA 98230; (604) 536-1886.

Reader Service Number 23

Supermicro family

Charles River Data Systems's Universe family of OEM super-microcomputers uses Motorola MC68000 and MC68020 microprocessors and the 32-bit VMEbus. The family includes the Universe/200, a six-slot, VME-backplane, 1.0-MIPS system based on the 68000 that supports up to 32 users; the Universe/400, 3.5-MIPS, 12-slot system that uses either Motorola microprocessor and supports 64 users; and the Universe/600, a departmental computer that supports 96 simultaneous users, 16M bytes of main memory, and from 40M bytes to 3.2G bytes of disk storage.

The systems run on UN/System V and the UNOS operating system and support the company's UniverseNet LAN controller and software. The systems also support 16 of the company's I/O processors, each of which contains its own 12.5-MHz 68000 and can handle up to 68 ports or be dedicated to support a high-speed line, SNA port, or multidrop connection.

The 1M-byte-memory /200 lists at $7995. Typical /400 systems are priced under $18,000 and /600s, under $35,000. OEM discounts are available.

Contact Charles River Data Systems, Inc., at 983 Concord Street, Framingham, MA 01701; (617) 626-1000.

Reader Service Number 24

The Universe/200 supermicro (right) from Charles River Data Systems can be dedicated by OEMs and technical end users to one application, while its companion systems, the /400 (left) and the /600 (center), can analyze and store graphic images, monitor a series of manufacturing operations, and handle multiple high-speed data communication lines.

TI introduces CMOS digital signal processor

Texas Instruments has announced its first CMOS DSP, the TMS320C10. This pin-for-pin and software-compatible version of the standard TMS32010 has been designed especially for power-sensitive applications. Its typical power dissipation is 125 mW, or 15 percent of the dissipation of the TMS32010. The highly pipelined architecture and programmable instruction set of the TMS320C10 executes five million instructions per second.

This 16/32-bit microcomputer also supports high-speed and computation-intensive applications, including telecommunications, consumer products, and computers and peripherals.

TMS320C10 provides a 16 × 16-bit hardware multiplier with 32-bit precision and communicates with off-chip memory at full speed. Other features are a 200-ns instruction cycle time, a 144-word on-chip data RAM, a zero to 16-bit barrel shifter, and a 16-bit bidirectional data bus with 40M-bps transfer rate.

The 40-pin, plastic dual-in-line TMS320C10 has a 100-piece price of $50. For further information contact Texas Instruments Semiconductor Group (SC-619), PO Box 809066, Dallas, TX 75380-9066; (800) 232-3200, ext. 700.

Reader Service Number 25

32-bit workstation aids inexperienced IC designers

The Application Specific Engine is a 32-bit super-microcomputer that permits engineers without IC design experience to produce working logic circuits. The ASE silicon compiler system, which also runs MS-DOS programs for the IBM PC AT, provides a 1280 × 800-pixel monochrome monitor with the GEM windowing system, 3M bytes of real memory, 16M bytes of virtual memory, and an 80M-byte hard disk.

Lattice Logic USA states that the system ensures correct ASIC design through its Chipsmith software, which prohibits designs that conflict with design rules supplied by foundries. Designs can be entered in two ways—by writing a circuit description in Model language or by using the system's schematic editor to convert a block diagram or schematic of the circuit into Model statements. The Model program is then compiled to form a netlist in an intermediate design language file.

The ASE simulator measures design changes for performance; the physical design subsystem automatically handles placement and routing.

ASE workstation prices begin at $15,995 for an entry-level model with schematic capture and simulator modules. A complete system with fixed array, chip generator, and standard cell modules costs $24,995.

For further information contact Lattice Logic USA, 3333 Bowers Avenue, Suite 199, Santa Clara, CA 95054; (408) 748-9797.

Reader Service Number 26

Tektronix offers PCB, VLSI simulators

The printed circuit board designer can now use a Tektronix Design Verification system to simulate concepts. DVS consists of four modules: logic simulation, fault simulation, test pattern generation, and hardware modeling.

A software interface allows the DVS to be used within the company's Designer's Database Schematic Capture environment. An integrated user interface provides an automatic netlist and allows designers to interact with DVS modules through the use of a pushbutton mouse and to go from simulation to layout using the same design schematic.

All modules are supported by device libraries of 2500 symbols and their simulation models. A HICHIP hardware modeling system simulates off-the-shelf or proprietary VLSI-based designs. HICHIP incorporates a VLSI device into the model library by physically connecting the device to a "personality" board and describing the external pin connections and their delay parameters with the modeling language.

Prices of the DVS modules start at $11,000. For more information contact Tektronix CAE Systems Division, 5302 Betsy Ross Drive, Santa Clara, CA 95054; (408) 727-1234, (800) 547-1512, or (800) 542-1877 in Oregon.

Reader Service Number 27

Operating system supports 60-PC network

A personal computer networking solution designed for MIS professionals and IBM-PC users is being launched in the US by Waterloo Microsystems Inc. The main features of Waterloo Port include icon-based user and DOS interfaces, on-the-fly configurability while the network is in use, multitasking capabilities, and a software architecture intended to free users from sacrificing workstations for dedicated server duty. This software feature enables network managers or users to distribute services such as printers, communications, and mass storage anywhere within the network's physical geography.

Port runs on a virtual RAM card, thus freeing up to 640K bytes of PC RAM for DOS applications such as spreadsheets and databases.

The base price of the Port network operating system software is $1695.

Waterloo Microsystems is based at 175 Columbia Street West, Waterloo, Ontario N2L 5Z5, Canada; (519) 884-3141.

Reader Service Number 28

Macintosh package changes outlines into charts

Living Videotext's MORE software for the Apple Macintosh changes outline structures into bullet charts or tree charts that can be displayed on the screen or output to a dot matrix or laser printer for paper or transparency copies.

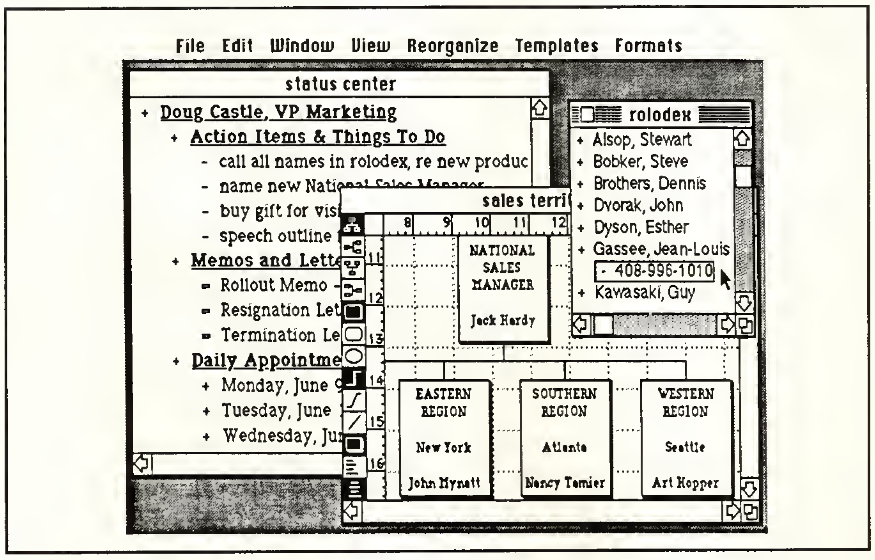
The outline processing options available include hoisting and dehoisting for more detailed looks at sections of an outline; cloning for creating linked cross-references throughout an outline; and pattern-matching with search, mark, and gather commands. In addition, users can include text and graphic windows within outlines, display multiple outlines concurrently, and select several heading and footer options.

MORE offers 32 installable outline templates to create personalized address files, business forms, questionnaires, or expense reports. Users can select borders, backgrounds, font styles, sizes, shading, and spacing. In addition, an outline math function is provided for totalling amounts listed in outline headlines.

The product comes on two disks, requires a Macintosh with 512K of memory or a Macintosh Plus with an external drive or hard disk, and sells for $295. It is not copy protected.

For further information contact Living Videotext, 2432 Charleston Road, Mountain View, CA 94043; (415) 964-6300.

Reader Service Number 29



status center

+ Doug Castle, VP Marketing
  + Action Items & Things To Do
    - call all names in rolodex, re new produc
    - name new Nati
    - buy gift for vis
    - speech outline
  + Memos and Lett
    - Rollout Memo
    - Resignation Let
    - Termination Le
  + Daily Appointme
    + Monday, June
    + Tuesday, June
    + Wednesday, Ju

rolodex

+ Alsop, Stewart
+ Bobker, Steve
+ Brothers, Dennis
+ Dvorak, John
+ Dyson, Esther
+ Gassee, Jean-Louis
  - 408-996-1010
+ Kawasaki, Guy

sales terri

NATIONAL SALES MANAGER Jack Hardy

EASTERN REGION New York John Hynatt

SOUTHERN REGION Atlanta Nancy Tomier

WESTERN REGION Seattle Art Hopper

MORE software from Living Videotext manages up to six outline windows at a time. The first of three windows displayed here holds a rolodex containing phone numbers that can be used with MORE's autodialer. A second window shows an outline that is transposed into the tree chart seen in window three. Tiling commands (not shown) automatically rearrange the windows so that all text is visible.

Amdek Corp.'s Model 1280 monochrome graphics display subsystem for IBM PCs and compatibles is suited for desktop publishing, computer-aided design, and engineering purposes in its 1280 x 800 mode. The 15-inch monitor and adapter also maintain compatibility with standard PC application software with 640 x 200 monochrome or CGA color emulation; colors can be mapped into four shades of intensity.

Model 1280 sells for $1595 from Amdek Corp. 2201 Lively Boulevard, Elk Grove Village, IL 60007; (312) 364-1180.

Reader Service Number 30

ASIC system produces RISC, DSP designs

An ASIC version of the Silicon Compilers Genesil Silicon Development System lets engineers design small, fast chips for digital signal processing, graphics, and reduced instruction set computers. According to the company, Version 86 has CMOS data-path enhancements that result in pipelined chip architectures and save routing to random logic blocks.

Version 86 is available on all Genesil Silicon Development Systems. Existing systems will be upgraded at no charge.

Silicon Compilers Incorporated is located at 2045 Hamilton Avenue, San Jose, CA 95125; (408) 371-2900.

Reader Service Number 31

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

High 192 Medium 193 Low 194

Letters *continued from page 4*

case had been constructed, and looked gorgeous, and some of the subsystems had been completed. The XR2206 chip that we were using as a voltage-controlled oscillator went from 20 Hz to 20 kHz linerarly but did not maintain the 0.1-percent distortion throughout the range. I hope my successor had luck with the project.

To digress a bit, when India's first rocket splashed into the sea and dumped its satellite payload instead of elevating it, the entire country was disappointed. But we were heartened by the words of Dr. Satish Dhawan, head of the Indian Space Research Organization. Commenting on the disaster, he said, and I quote, "We have slipped, but we haven't fallen flat on our face." Eventually, India did successfully launch a communications satellite.

Chandan Sen
Raleigh, North Carolina

CRC revisited

To the Editor:

I was very intrigued about the speed with which the 8086 CRC routine (August 1985, pp. 4, 99, *IEEE Micro*) would run and decided to implement it in 68000 code. However the prospect of entering the lookup table was not very appealing, not only from an entry-time standpoint but especially from the fact that the probability of a typographical error would be high.

Below are two macros that will generate the lookup table for CRC-16 using Motorola format macros. Originally the CRC macro was written recursively, but the assembler would not support a macro nested 256 deep. Hence the macro CRC is called 256 times to generate the 256 lookup values. This method will work as well using any macro assembler and on any CRC format via Perez' algorithm (June 1983, pp. 40-50, *IEEE Micro*).

Some notes on Motorola macros: !! is 'exclusive or,' < < is 'shift left,' DC is 'define constant word,' IFNE is 'if operand not equal to zero,' & is 'logical and,' and $ indicates hexidecimal.

Figure 1 (from the August 1985 *IEEE Micro*) is the CRC-16 polynomial from the exclusive or of the four words. The lookup table is the exclusive or of the last three words. The following code written in C does the remaining exclusive or.

MS	bit														
0	0	0	0	0	0	0	0	C16	C15	C14	C13	C12	C11	C10	C9
Px	Px	0	0	0	0	0	0	0	0	0	0	0	0	0	Px
0	0	X8	X7	X6	X5	X4	X3	X2	X1	0	0	0	0	0	0
0	X8	X7	X6	X5	X4	X3	X2	X1	0	0	0	0	0	0	0

Notes:

Ci are the bits of the previous CRC word value.
Xi are the individual bits of the exclusive or of the data byte and the log byte of the old CRC.
Px is the parity of the X byte.

Figure 1. The CRC-16 polynomial is created by the modulo 2 addition of four words.

Table 1.
Lookup table.

```
* CRC macro generates lookup table.
 CRC       MACRO
 PAR       SET      0                         * init set bit counter
                                              *    for parity macro.
           PARITY   CRCSEED                   * determine parity
           IFNE     PAR&1                     * check if parity odd
                                              *    (least sig bit set)
           DC       $C001!!(CRCSEED<<6)!!(CRCSEED<<7)
                                              * do modulo 2 add
           ELSE                               * else if parity even
           DC       (CRCSEED<<6)!!(CRCSEED<<7)
           ENDC                               * end conditional
 CRCSEED   SET      CRCSEED+1                 * get ready for next CRC
           ENDM                               * end macro
* Parity macro determines the parity of passed number.

 PARITY    MACRO    P1
           IFNE     P1                        * any set bits left?
           IFNE     P1&1                      * check least sig bit
 PAR       SET      PAR+1                     * count only bits that are set
           ENDC                               * end conditional
           PARITY   P1>>1                     * do next bit
           ENDC                               * end conditional
           ENDM                               * end macro
* CRC lookup table

             XDEF         CRCTAB              * make CRCTAB global

 CRCSEED     SET          0                   * start 0-255 CRC counter
                                                (index)

 CRCTAB:                                    * call CRC macro 256 times
             CRC
             CRC
             CRC
              .
              .
              .
              .
```

```
crc=0;  /* init the crc */
/* lngth is the length of the message */
/* p_msg is a pointer to the message */
for (;lngth; lngth--)
   crc=(crc >> 8) ^ crctab
   [(crc^ *p_msg++) & 0xff];
```

Rick Bronson
Milwaukee, Wisconsin

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

High 183 Medium 184 Low 185

Advertiser Index—August 1986

Addison-Wesley	7
C.A.C.I.	5
Casio, Inc.	Cover III
IEEE COMPUTER GRAPHICS AND APPLICATIONS Subscription	34
MIMI '87	Cover II
Omega Engineering, Inc.	Cover IV

FOR DISPLAY ADVERTISING INFORMATION CONTACT

Southern California and Mountain States: Richard C. Faust Company, 24050 Madison Street, Suite 100, Torrance, CA 90505; (213) 373-9604.
Northern California and Pacific Northwest: Don Farris Company, 161 W. 25th Ave., #102B, San Mateo, CA 94403; (415) 349-2222.
Jack Vance, P.O. Box 3205, Saratoga, CA 95070; (408) 741-0354.
East Coast: Hart Associates, Inc., P.O. Box 339, 42 Lake Blvd., Matawan, NJ 07747; (201) 583-8500.
New England: Arpin Associates, P.O. Box 227, Weston, MA 02193; (617) 899-5613.
George Watts, III, 4 Conifer Dr., Wilbraham, MA 01095; (413) 596-4747.
Midwest: Thomas Knorr, Knorr MicroMedia, Inc., 333 North Michigan Ave., Chicago, IL 60601; (312) 726-2633.
Southwest: The House Company, 3000 Weslayan, Suite 345, Houston, TX 77027; (713) 622-2868.

Advertising Director: Mike Koehler, *IEEE MICRO*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-3240, 821-8380.

For production information, conference or classified advertising contact Carole L. Porter, *IEEE MICRO*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-1140.

PRODUCT INDEX

	RS#	Page#
Addison-Wesley	1	7
Amdek Corporation	30	78
Casio, Inc.	2	Cover III
Charles River Data Systems	24	77
Latice Logic USA	26	77
Living Videotext	29	78
Logic Automation, Inc.	20	76
Motorola	22	76
NCR Microelectronics	21	76
Omega Engineering, Inc.	3	Cover IV
Pacific Microcircuits Ltd.	23	76
Silicon Compilers Incorporated	31	78
Texas Instruments Semiconductor Group	25	77
Tektronix CAE Systems Division	27	77
Waterloo Microsystems	28	78

Moving?

MAIL TO:
IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

PLEASE NOTIFY US 4 WEEKS IN ADVANCE

Name (Please Print)

New Address

City State/Country Zip

ATTACH LABEL HERE

- This notice of address change will apply to all IEEE publications to which you subscribe.
- List new address above.
- If you have a question about your subscription, place label here and clip this form to your letter.

IEEE MICRO

For further information on advertised products, new products, or literature, fill out the **Reader Service Card** (top). Circle the number on the RS Card that corresponds to the number of the item for which you would like more information.

To indicate your interest in an article or department, fill out the **Reader Interest Card** (bottom). Circle the number on the RI Card that corresponds to the level of interest given in the Reader Interest Survey at the end of the article or department.

Please print or type your name and address.

READER SERVICE CARD

IEEE MICRO INFORMATION ABOUT PRODUCTS

Void after February 28, 1987 **8/86**

Name ______

Company ______

Address ______

City ______

State ______

Zip ______

Country ______

Title ______

Telephone number ( ) ______

PRODUCTS PURCHASED OR SPECIFIED	FOR JOB	FOR HOBBY
Computers	80	81
Peripherals	82	83
Data communications equip.	84	85
Memories, components	86	87
Software and services	88	89
Publications	90	91
Other	92	93

94 Please send me information on advertising in *IEEE Micro.*

95 Please send me the IEEE-CS *Publications Catalog.*

96 Please send me an IEEE Fellow nomination form.

97 Please send me an IEEE Computer Society membership application.

98 Please send me an *IEEE Micro* author's guide.

Product announcements, and products for review, should be sent to Marie English, Managing Editor, *IEEE Micro*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.

Send more information on numbered items:

1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61	65	69	73	77	81	85	89	93	97
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62	66	70	74	78	82	86	90	94	98
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63	67	71	75	79	83	87	91	95	99
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80	84	88	92	96	100

READER INTEREST CARD

IEEE MICRO EDITORIAL RESPONSE

8/86

Name ______

Company ______

Address ______

City ______

State ______

Zip ______

Country ______

Title ______

Telephone number ( ) ______

Reader Interest Survey:

101	116	131	146	161	176	191
102	117	132	147	162	177	192
103	118	133	148	163	178	193
104	119	134	149	164	179	194
105	120	135	150	165	180	195
106	121	136	151	166	181	196
107	122	137	152	167	182	197
108	123	138	153	168	183	198
109	124	139	154	169	184	199
110	125	140	155	170	185	200
111	126	141	156	171	186	201
112	127	142	157	172	187	202
113	128	143	158	173	188	203
114	129	144	159	174	189	204
115	130	145	160	175	190	205

Comments:

Continue comments on other side

This PO box for reader
service cards only.

PLACE
STAMP
HERE

IEEE MICRO

Reader Service Inquiries
Box 24168
Los Angeles, CA 90024
USA

Comments on articles and other editorial matter:

I liked ____________________

I disliked ____________________

I would like ____________________

For Reader Interest Survey, see other side

PLACE
STAMP
HERE

IEEE MICRO

10662 Los Vaqueros Circle
Los Alamitos, CA 90720-2578
USA

From micro to mainframe, Casio's got the hardware for your software.

At first glance, it may look like an ordinary calculator, but the Casio solar CM-100 is anything but. It's an extraordinary software tool that's as useful in programming an Apple™ as it is a mainframe IBM.™

The key to the CM-100's incredible flexibility is Casio's adjustable bit-size selector which can be set to suit any size computer up to 32 bits. And its block display which can, by scrolling blocks of 8 digits at a time, display up to a 32 bit word.

But there's much more to this pocket-size powerhouse. It can do base conversions from binary/octal/decimal/hexadecimal modes and can store in its memory numbers in any base. It also has Shift, Rotate, Arithmetic Shift and Boolean functions that include AND, OR, XOR and NOT.

Perhaps what is most extraordinary about the CM-100 though, is not how much it can do, but how little it costs to do it. The CM-100 is the only calculator that'll let you do all your software figuring for less than you'd figure to pay for an average ($25.00) textbook.

The more you work with computers—whatever their size—the more you need a CM-100. Whether you're a student or professional, it's the one piece of hardware that will make designing your software easier. For more information, call 1 (800) 553-3338 X32.

Apple and IBM are trademarks of the Apple and IBM Corporations.

CASIO®
Where miracles never cease

Casio, Inc. Consumer Products Division: 15 Gardner Road, Fairfield, N.J. 07006

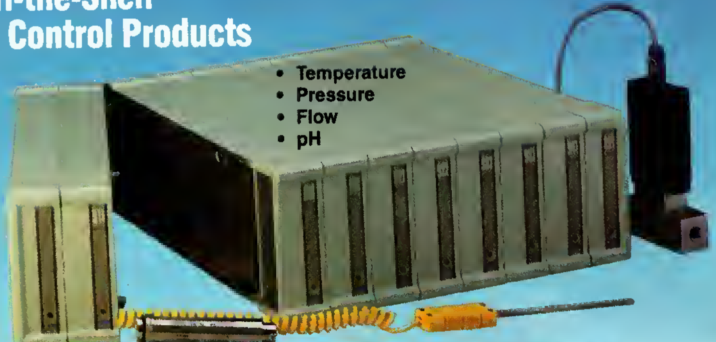
Reader Service Number 2

GET THE NEW! FREE! 1986 HANDBOOKS NOW!

✓ Thousands of Off-the-Shelf Measurement & Control Products

- Temperature
- Pressure
- Flow
- pH

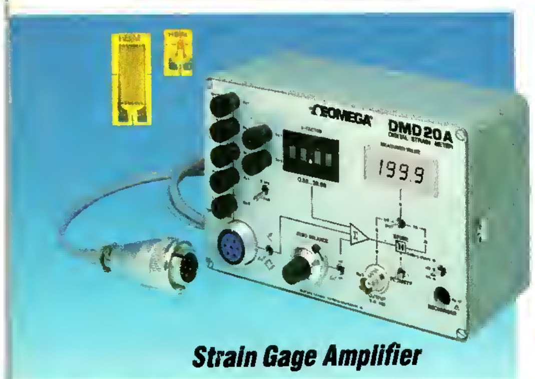
Precision X-Y and Y-t Recorders


Intelligent Signal Conditioning Computer Interfaces

pH Instrumentation

Totalize Flow

Data Log


Strain Gage Amplifier

Pressure Transducers

Process Readout

1986
OMEGA
COMPLETE PRESSURE AND STRAIN MEASUREMENT HANDBOOK AND ENCYCLOPEDIA
1986
OMEGA
COMPLETE TEMPERATURE MEASUREMENT HANDBOOK AND ENCYCLOPEDIA
OMEGA
(203) 359-1660

- *Everything at Your Fingertips!*
- *Off-the-Shelf for Immediate Delivery!*
- *All Prices Included in Every Handbook!*
- *Largest Selection of Temperature, Pressure, Flow & pH Products in the World!*

For placing orders, dial (203) 359-1660!

® OMEGA ENGINEERING, INC.

(203) 359-RUSH

One Omega Drive, Box 4047, Stamford, CT 06907
Telex 996404 Cable OMEGA FAX (203) 359-7700

1986
OMEGA
COMPLETE FLOW AND LEVEL MEASUREMENT HANDBOOK AND ENCYCLOPEDIA
1986
OMEGA
COMPLETE pH AND CONDUCTIVITY MEASUREMENT HANDBOOK AND ENCYCLOPEDIA
pH
OMEGA
(203) 359-1660

Circle Reader Service Number
to Receive Your NEW! FREE! Handbooks.

Reader Service Number 3